Scalable Parallel Astrophysical Codes for Exascale

Code release (alpha)

Deliverable number: D1.3

Version 1.0/1.0

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:	Pranab J Deka (KUL)
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual Delivery Date:	30.06.2024
Actual Delivery Date	01.07.2024
Number of pages:	38
Keywords:	modules and kernels, optimisation, scalability, GPU porting
Authors:	Benoît Commerçon – CNRS 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:	Kristian Kadlubiak (IT4I) Marc Sergent (Eviden)

History of Changes

Release	Date	Author, Organisation	Description of changes
0.1	08.04.2024	Gino Perna (ENGINSOFT), Luca Tornatore (INAF)	Started the document
0.2	18.06.2024	Marco Rossazza, Andrea Mignone, Stefano Truzzi (UNITO)	PLUTO section
0.3	19.06.2024	Robert Wissing (UiO)	ChaNGa section
0.4	21.06.2024	Benoît Commerçon (CNRS) Georgios Doulis, Khalil Pierre (GUF)	RAMSES section FIL and BHAC sections
0.5	22.06.2024	Pranab J Deka (KUL)	iPic3D section
0.6	23.06.2024	Luca Tornatore (INAF), Geray Karademir (LMU)	OpenGADGET section
0.7	27.06.2024	Kristian Kadlubiak (IT4I), Marc Sergent (Eviden)	Proofreading
1.0	01.07.2024	Pranab J Deka (KUL)	Final version assembled

Scalable Parallel Astrophysical Codes for Exascale

DISCLAIMER

Executive Summary

This document outlines the current achievements and ongoing efforts within the SPACE CoE. The primary focus is on optimising and porting kernels or modules of the codes involved in SPACE CoE to GPU architectures and adapting them for efficient use on EuroHPC JU clusters. These advancements are crucial for addressing the increasing computational demands of large-scale, long-duration astrophysical simulations.

The document is structured to provide detailed updates for each code involved in the project, covering the current status of GPU porting, optimisations, CI/CD implementation, and performance on EuroHPC JU clusters. Let us also note that all of the seven codes now have an open-source/public repository, allowing the progress within the code to be tracked in the respective repositories. The combined efforts in these areas demonstrate the project's commitment to advancing HPC capabilities and supporting the computational needs of astrophysical research.

Contents

List of Figures

List of Tables

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
DCGP	Data Centric General Partition
DD	Domain Decomposition
ET	Einstein Toolkit
EuroHPC JU	European High Performance Computing Joint Undertaking
GRMHD	General Relativistic Magnetohydrodynamics
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
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

1 Introduction

The advancement of high-performance computing High Performance Computing (HPC) applications is pivotal for the progress of scientific research, particularly in fields demanding substantial computational power such as astrophysics. This deliverable outlines the efforts undertaken within the Scalable Parallel Astrophysical Codes for Exascale (SPACE) Centre of Excellence (CoE) to optimize and port several key HPC applications to GPU architectures and adapt them for efficient use on European High Performance Computing Joint Undertaking (EuroHPC JU) clusters. These initiatives are critical in addressing the growing computational demand of large-scale long-duration complex simulations.

GPUs, with their superior parallel processing capabilities, offer a significant advantage over traditional CPU-based computations. This deliverable focuses on the recent advancements of porting kernels and modules to GPUs, optimising to achieve further speedups, and in some cases, the scaling results on multiple GPUs. It is worth noting that some applications are in the initial stages of porting and optimising kernels on GPUs and they have presented a gist of the planned work in the upcoming months.

Another central aspect of this deliverable is the implementation of Continuous Integration (CI)/Continuous Deployment (CD) pipelines. These pipelines facilitate automated testing, integration, and deployment of code. CI/CD approach ensures that code updates are seamlessly incorporated and rigorously tested, maintaining high standards of code quality and reliability. This collaborative framework not only streamlines development processes but also fosters a dynamic and responsive development environment.

Finally, this deliverable also reports on progress in adapting these applications to EuroHPC JU clusters. By ensuring that HPC applications, within the framework of SPACE CoE, are optimised for these clusters, we aim to make advanced computational resources more accessible to researchers, thus broadening the scope and impact of scientific investigations.

The following chapters are organised as one per code, where each code owner presents the current status of GPU porting and optimisations involved in such, CI/CD implementation, and running on different EuroHPC JU clusters. In summary, this deliverable details the progress made across various applications within the SPACE project. The combined efforts in GPU porting, kernel optimisations, and CI/CD implementation demonstrate our commitment to advancing HPC capabilities.

2 OpenGADGET

2.1 Introduction

OpenGADGET is a Lagrangian code that solves the Vlasov-Poisson equations in a cosmological expanding framework. 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. In addition, the Smoothed Particle Hydrodynamics (SPH) algorithm is used to solve the baryonic hydrodynamics. A broader description of the code is available in §2 of D1.2.

2.2 CI/CD implementation

In the past, OpenGADGET used a private SVN repository combined with a Jenkins instance for continuous integration. In April of this year, the developer repository of OpenGADGET was transitioned to a GitLab repository hosted by Leibniz-Rechenzentrum (LRZ).

As the repository at the LRZ GitLab is private, we have mirrored the main_development branch to the IT4I GitLab (`https://code.it4i.cz/space_coe/opengadget3`) for everyone in the CoE to have access to the code as well as to see the current development. The public repository for the publication of OpenGADGET will be hosted at the LRZ as well (`https://gitlab.lrz.de/MAGNETICUM/OpenGADGET3`). Since the public version of the code is not ready yet it is currently empty. The publication of a functional public version including documentation, etc. is aimed to be done by December 2024.

The implemented tests themselves can be split into two main categories. Firstly there are regular tests, which run at every merge request as well as at each night during the week. Secondly, there are additional tests, which are run less frequently due to their larger computational demand. Both types of tests are described in more detail in the following section.

In general, our CI/CD pipeline is set up so that it automatically runs on these occasions:

- when scheduled: one pipeline at midnight each day.
- on merge request: all tests are executed for each merge request and have to be satisfied for the merge request to be accepted.
- on demand: the pipeline (or parts of the pipeline) can be run by each developer on any branch to test their recent developments at any time.

This pipeline is running on a small machine with 10 CPU cores at the LRZ in Garching near Munich and can also be run on the machines by IT4I if necessary. For now, we have not yet run the pipeline on additional JU machines, but are going to add this capability in the future (see Sec. 2.2).

Regular tests

The regular tests for OpenGADGET consist of tests with a total runtime of less than 10 minutes. The aim of these tests is firstly to catch compilation errors and secondly to check the basic routines such as the SPH and gravity solver. The short runtime allows us to run these tests for every merge to catch potential bugs early during the development process. The pipeline for the regular tests consists of multiple stages, which can be seen in Fig. 1.

- Indent: checking for indentation and coding style of changed files. This is only done for merge requests and is currently allowed to fail as we have decided to change our code format just recently and aim to do the formatting of the code at a specific time for all code files at once. For the format we have decided to switch from a custom gnu-indent style to clang-format and the "Microsoft" style.
- Config tests: in this test, six different configurations are used to build the code.
- Unit test: Here the integrity for all equations of state classes, the Riemann solvers, Matrix operations, flux solvers, fluid vector classes and vector operations are verified.
- SPH tests: Running a Sod shock tube problem `https://en.wikipedia.org/wiki/Sod_Shock_Tube` , a Sedov blast wave problem `https://en.wikipedia.org/wiki/Taylor%E2%80%93von_Neumann%E2%80%93Sedov_blast_wave` and a Sound wave test and comparing the result with their analytic solution using SPH.

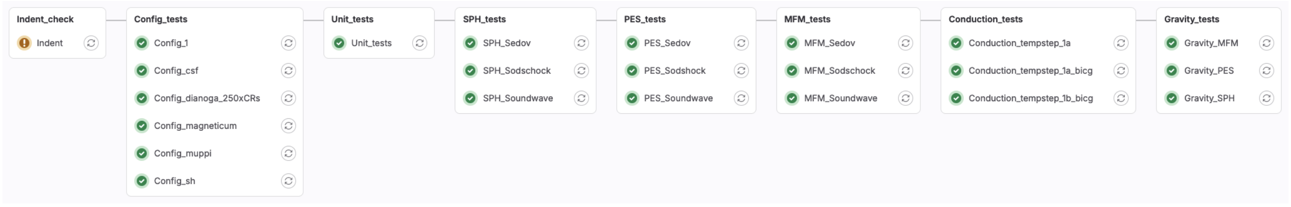


Figure 1: This figure displays all basic tests of the CI/CD pipeline for OpenGADGET. It is intended that all of these tests have to pass before merging on the main_development branch.

- PES tests: Same tests as for SPH, but with pressure entropy SPH (PES).
- MFM tests: Same tests as for SPH, but with a Meshless-Finite-Mass implementation.
- Conduction tests: Testing the SPH conduction at different timesteps.
- Gravity tests: in this test, the calculation of the gravity with all three hydro implementations is tested.

Additional tests

In addition to the relatively short test described above, we implemented two larger test cases to check the capability to run a full simulation with full physics as well as the physical output of the code.

- Galaxy simulation: In this test, a full cosmological zoom-in simulation is executed to check the global properties and results of the simulation (∼90 CPUh).
- Magneto hydrodynamics test: This test checks the magnetic fields properties by running a Ryu and Jones 5A test [2] (∼30 CPUh)

Each of these tests consists of two jobs: one running the simulation and a second to evaluate the output. The focus of these tests is mainly the output of the simulation.

These tests provide the opportunity to test additional components such as the ability to run the simulation for the full cosmic time, its runtime, its restart capabilities and its OpenMP and Message Passing Interface (MPI) implementations as these tests are executed in parallel on 8 CPU cores. This allows us to check for potential flaws in the code from a different point of view, even in the absence of hard errors. Due to their significantly larger runtime of up to 13 hours on the current machine, these tests are not suited for daily tests. Instead, an extended pipeline is executed on the weekends including these tests.

Future aims

While already having a substantial pipeline in place, we are going to add a few more tests to get an even better assessment of the current status of the code. The first step here is to add additional tests for currently untracked modules and config switches and to implement a measure for the code coverage. For this automated testing of code coverage, we are currently searching for the best solution to our needs. As the porting to GPU is a critical part of the code we aim to translate some of the tests to be executed on GPU as well. The major issue for this task is the high demand for GPUs in general as for our test we would require to have reliable access to at least one GPU. The final aim would be to verify the build of the code at different systems, such as the SuperMUC-NG at the LRZ, Leonardo Booster and Data Centric General Partition (DCGP) at CINECA, Karolina and Barbora at IT4I, and finally at JUPITER at JSC. As this will require the installation of GitLab runners on each of these machines as well as triggering automated jobs this might become difficult due to the different safety requirements of the different computation centres.

2.3 Running on EuroHPC JU Clusters

At the time of writing this document, the code has been compiled and ran on the the CPU partitions of the EuroHPC JU clusters Karolina and Barbora (see CPU benchmarks in D2.1 and D2.2) and the scaling tests for the GPU implementation were done on Leonardo's Booster partition. In addition, compilation was done on LUMI-C as well and we are currently applying for access to the rest of the EuroHPC JU clusters via an activity by WP4.

2.4 GPU porting and Kernel Optimisations

2.4.1 GPU porting

For the GPU porting of OpenGADGET, the modules with the largest contribution to the total runtime have been chosen: the tree walk and the Barnes&Hut oct-tree.

Current status

In the larger framework of the code, the tree walk is called at multiple stages as the tree conveys information about the spatial locality of the particles. This means that the computation of each physical process in which a particle impacts additional particles relies on the tree walk to determine the neighbourhood of the active particle. Or, conversely, to determine the neighbourhood that is affecting a target particle.

The current GPU implementation is based on porting the computationally heavy part of the tree-walk to the GPU while keeping the global structure. Consequently, only the local contributions are computed on the device, while the communication and all other aspects remain on the host. For this, a new data structure is created, which only contains the information about the available particles, which is then communicated to the device and back after the calculation is finished.

Using this approach allows us to implement the GPU tree walk by simply annotating CPU functions. The benefit of this approach is that the same code is used on the host as well as on the device, so updates and bug fixes on the tree walk itself are immediately available in both code versions.

Summarising, the following modules have been ported onto GPU in the current version:

1. **Gravity calculation with tree** The tree-walk and the Barnes&Hut algorithm have been offloaded, adding necessary directives to the existing code. The entire tree array is uploaded to GPU, as well as the subset of particles' data that are of interest. For this purpose, a specific array is built, extracting the relevant quantities from the larger structure that contains all the particles' data.

2. **SPH Hydrodynamics** This module consists of two separate tree walks. *The first one* iteratively defines the SPH neighbourhood of a target particle, i.e. the sphere radius that encompasses N_{SPH} neighbours. Starting from a guess, the tree exploration is iterated with a different radius R_{SPH} until the number of gas particles found within R_{SPH} is correct. *The second walk* iterates over the list of N_{SPH} neighbours to estimate the hydrodynamical quantities and solve the hydrodynamics using the SPH approach.

3. **Thermal conduction** This process amounts to energy transport and is solved with a conjugate gradient. As well as Hydrodynamics, it relies on a tree-walk to estimate the needed physical quantities in the hydrodynamical neighbourhood within R_{SPH}.

NOTE: the GPU implementation has started before the project SPACE, and is due to the work of Dr. Antonio Ragagnin and LMU's staff. An important part of the work has been conducted during the first 18 months, either by LMU within the SPACE project in collaboration with Dr. Ragagnin. Namely, several optimisations on the Gravity part (mostly the overlapping of computation and communication) and the porting of the hydrodynamics loop.

Scaling of the current GPU offloading

In the frame of SPACE, thanks to the collaboration with CINECA, we have been able to perform a large number of tests and assess both the strong and weak scaling of the code using up to 3072 nodes (90% of Leonardo's Booster partition). However, due to the limitations of the available computational resources, we have not been able to profile and assess the scaling of entire simulations but just of some initial time-steps.

We have chosen to simulate the computational boxes included in our scientific cases, explicitly generated for this purpose (the entire set of initial conditions amounts to 3TB). However, we do not have evolved realisations of these cases, since they are computationally very expensive. Then, we simulated only the beginning of the Universe's history at high redshift (we have generated the boxes at $z = 50$). The density distribution at that epoch is very homogeneous, and that is reflected in a non-deep tree and in a very non-clustered distribution of particles; this is a particularly favourable situation for our algorithm, and that is to be accounted for. In a second scaling assessment campaign, we will be able to use evolved boxes or single-objects that come from the results of other projects. For the strong scaling, we used boxes with $1024^3, 2048^3$, and 4096^3 particles spanning the whole amount of nodes (see Fig. 2) since a case that fits the lower end would lead to small amounts of

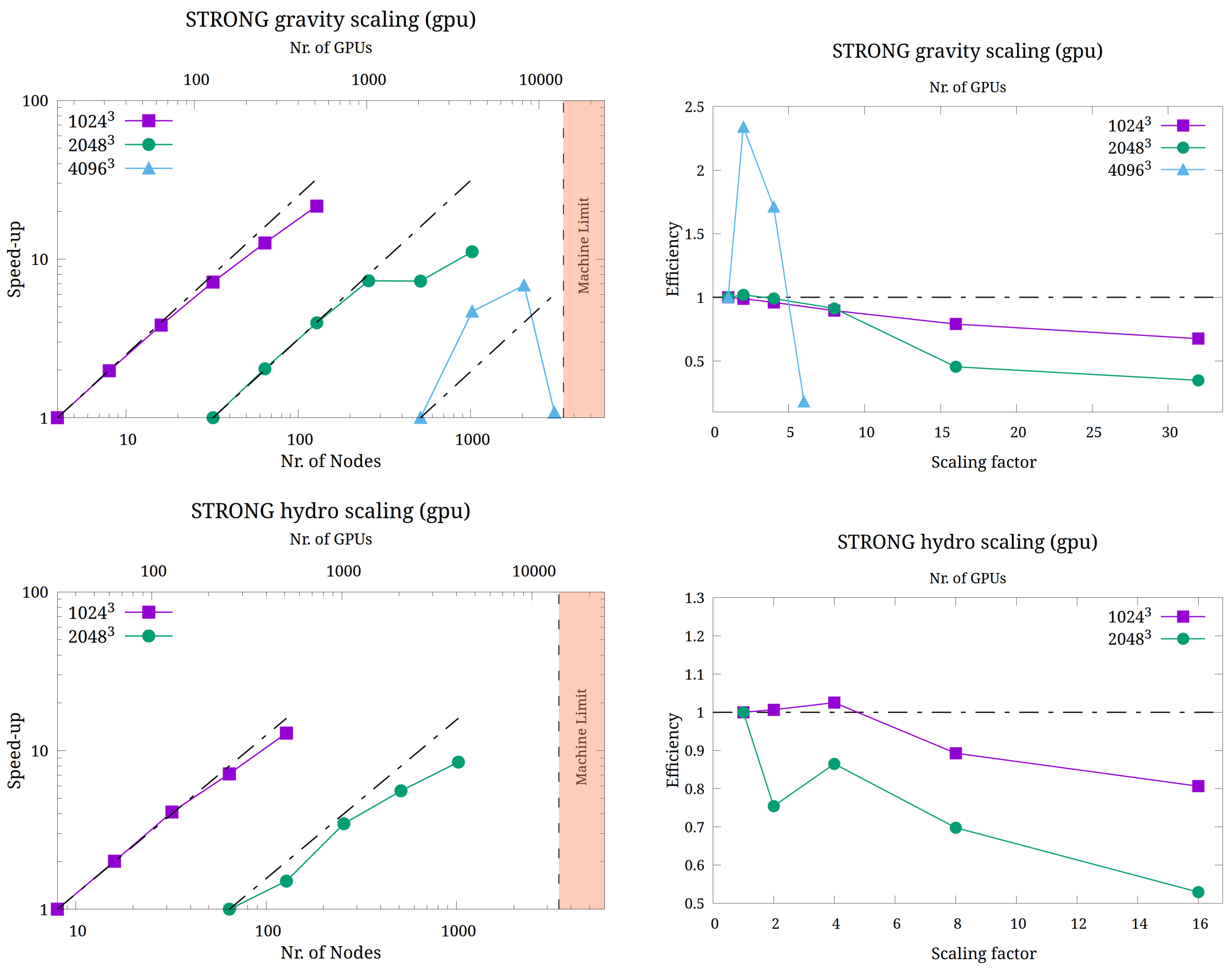


Figure 2: The complete results of **strong scaling** test for both the Gravity-only (**upper panel**) and Hydro (**bottom panel**) cases, from 4 up to **3072 nodes** on Leonardo Booster. The left column shows the number of nodes vs speed-up and the left node the scaling factor vs efficiency. The **Black dashed** lines represent ideal speedup. In all the runs, the **MPI/OpenMP** configuration is such that one MPI task was running per GPU, while all the 32 available CPU cores were filled with OpenMP threads, which amounts to 4 MPI tasks/node and 8 threads/task. Note that some issues arose in domain decomposition with the 2048^3 case after 512 nodes, and a network problem has invalidated the 3072 nodes run of the 4096^3 case.

data per process at the high end, becoming artificially communication-dominated. On the other hand, a case that could run on the whole machine could not fit in the memory of a small sub-set of nodes. The sequence $(1024^3, 1296^3, 1632^3, 2048^3, 4096^3)$ has been used for weak scaling tests.

In Fig. 2 we show the results of the strong scaling test for both the gravity-only (upper panel) and the hydrodynamics (lower panel) test cases. The speed-up and the parallel efficiency are on the left and right columns, respectively. While the **1024^3 case** behaves smoothly, slowly decreasing in efficiency which remains $> 80\%$ for a $32\times$ scaling factor, the **2048^3 case** shows a sudden decrease beyond $16\times$ for both the Gravity and the Hydro cases. Conversely, the **4096^3 case** (which we run only for the Gravity-only case due to the limits on the available computational time) exhibits a super-linear speed-up for $2\times$ and $4\times$, while a malfunction in the network led to a sudden increase in the communication time for $8\times$ (3072 nodes). We plan to repeat this run as soon as the whole machine is available for large tests.

As a general conclusion, **the current GPU implementation offers a viable opportunity to run a simulation with a net gain of about 3**$\times$, which is a very significant result given that state-of-the-art simulations require a computational effort of the order of 10^7 core-hours.

An updated GPU implementation

In a many-body code, such as OpenGADGET3, we use the Barnes & Hut algorithm to estimate the force F_i, reducing the algorithmic complexity to $O(N\texttt{log}N)$ (compared to $O(N^2)$ via direct summation): relying on the tree structure to distinguish between "close particles" that are summed directly as in the former equation, and "clusters of distant particles" that are considered as an effective unique particle whose contribution is summed to F_i.

The Barnes & Hut algorithm, in its "classical" formulation, although reducing the algorithmic complexity, proves to be inefficient on modern architectures and particularly on GPUs since it leads to a large amount of thread divergence and locking. At the time of writing, we are developing an alternative formulation where we aim to retain the overall $N\texttt{log}N$ scaling but should be more efficient on GPUs (and, possibly, on modern CPUs, too).

Our **new strategy** is based upon the following pillars:

- Grouping particles; performing the tree walk & evaluation from the same position for a group of spatially closed particles while assigning same-group particles to same-warp threads allows (i) to shrink the number of tree walks (as much as the "grouping factor") and (ii) to synchronize the ops for threads on the same warp (when assigning a target particle per thread).
- Avoiding the tree walk in an "inner radius"; inside a given inner radius, we avoid evaluating nodes and immediately sum up, with direct summation, all the particles of nodes that are encompassed within that radius. That is feasible thanks to the memory arrangement of the tree nodes and the indexed traversal already implemented in OpenGADGET.

As a first trial, we opted not to keep the particle structures close in memory. The actual loop that updates particles' force value gets through the list of active particles and not the list of particles (note: we remind the reader that the particles are assigned with individual time-steps within a hierarchy of power-of-two decomposition of the timeline. Particles that are subject to more intense accelerations are integrated more often. Hence, at every code step, only a subset of particles is to be updated). Hence, we opted for

- traversing the list of active particles, propagating the information through the hierarchy of tree nodes so that every node knows how many active particles it contains;
- individuating the target nodes, i.e. the group of particles that will be treated collectively, following the criteria explained above;
- partitioning the list of active particles so that the active particles that belong to the same target node are subsequent in the list, which is a crucial step. After that, the loop which offloads the new routine for the modified Barnes & Hut could just assign subsequent active particles to threads in the same warp.

Most of the code is already written at the time of this writing; we estimate that we will be able to run the first test by September 2024.

3 PLUTO

3.1 Introduction

PLUTO (https://plutocode.ph.unito.it) is a finite volume grid code solving hyperbolic and parabolic conservation laws on a static grid or adaptive grid. The code is mainly intended for fluid and plasma physics application in an astrophysical context, with a special attention high-energy astrophysical phenomena.

Within the SPACE proposal, only the static grid version of the code is being upgraded to GPU leaving Adaptive Mesh Refinement (AMR) as a future task. The newly developed GPU version (gPLUTO) shares the same underlying philosophy of its predecessor (PLUTO) and a high-level description of the code and the main algorithms may be found in §3 of D1.2. The new exa-scale version of the code (gPLUTO) with full support for GPU, and currently under development within the SPACE CoE, is available as a GIT repository at https://gitlab.com/PLUTO-code/gPLUTO.

3.2 CI/CD implementation

The GPU porting of the code started with a simplified mini-app ($\sim 1,000$ lines), entirely rewritten from scratch while retaining the original code structure and subsequently adapted and extended for optimal exploitation and integration with OpenACC programming model. The code has grown ever since and it counted - at the end of the first year of the project - approximately $\sim 40,000$ lines, 70% of which have been completely rewritten (only I/O retains its original structure).

gPLUTO is being developed by a restricted number of contributors addressing different and specific activities. While activities may be interchanged within this restricted group, they belong to either one of these tasks: i) GPU kernel and module implementation & optimisation, ii) numerical benchmark porting and comparison, iii) readability, code quality improvements and general maintenance. More closely:

- Task i) focuses on porting modules from the previous CPU version (PLUTO) to the new version (gPLUTO). During this task we also have the opportunity to undertake a revision process for an almost 20 years old code, by removing unused or useless algorithms, increase user-friendliness, enhance parallel performance.
- Task ii) is essential in order to ensure that previous results can be faithfully reproduced. This is part of the quality assurance process and it is described later.
- Task iii) ensures overall consistency and coherence in terms of code syntax (indentation, comments, C++ naming convention, array allocation, macro employment, and so forth.).

GitLab is our chosen continuous integration application, and the repository may be found at https://gitlab.com/PLUTO-code/gPLUTO. Changes are committed several times a week, coordinated by an almost day to day interaction between <u>all</u> the developers, in order to avoid unwanted concurrent operations on the same branch or to simply inform others about recent bug fixes or kernel modifications. Modifications achieved in a few days (or less) are done directly on the main branch while more demanding efforts with specific feature require separate branch. Major changes that can affect more than one module or significantly compromise backward compatibility, however, are first discussed and brought to the attention of the main code developer.

Changes committed to the main branch are successfully validated only after an extensive suite of numerical benchmarks passes without errors. This activity - which has also been used in the CPU version of the code for more then 15 years - is automated by a Python script which compiles and runs different test problems and check that newly produced log files match the fiducial ones, previously produced by the previous CPU code version (PLUTO). Log files contain information like the Mach number, maximum / minimum value of primitive quantities, time step, and so forth, that are sensitive to arithmetic precision. Failure in replicating a fiducial log file demands a more accurate debug activity that can last from just a few hours to even a week (depending on the case). When the differences may - beyond any reasonable doubt - be ascribed to accumulated machine-precision errors produced by the different compilers (e.g. `gcc` vs `g++` vs `nvc` or `nvc_acc`) or a different operation order due to the optimised kernel implementation, the corresponding reference log file is replaced with the newer one. This typically occurs for the relativistic fluid modules where the number of operation counts, including also square roots and root-finder algorithms, is larger. Similarly, problems with chaotic behaviour (e.g. turbulence, strongly unstable systems) lead to similar problems.

Notice that for every test problems, we employ several *configurations* based on different combination of algorithms; for each configuration, we further adopt alternative *builds* (that is, using different compilers, number of processors, targeting either CPU / GPU) can be arbitrary defined, in order to check (and enhance) portability. At present, this suite includes 33 tests with several configurations and serial / parallel builds, for a total of ~ 320 test cases. Finally, in order to avoid architecture-induced log discrepancy, tests are run on a dedicated workstation equipped with 2 CPUs Intel Xeon 26-Core 5320 2.2 Ghz and 2 NVIDIA RTX A4500 graphics cards.

3.3 Running on EuroHPC JU Clusters

At the time of writing this document, the code has been compiled and can run on several EuroHPC JU clusters. We recently compiled, executed, and conducted scaling tests (Fig. 3) on both Leonardo Booster and Leonardo Data Centric partitions. Additionally, we have successfully compiled and ran the code on Karolina (compute nodes with GPU accelerators) and on the CPU nodes of LUMI-C. Moreover, our proposal for EuroHPC Development Access Call was accepted for all of the requested HPC clusters. We successfully ran on VEGA CPU and GPU partitions. Currently, we are working on the access procedures for MeluXina (CPU and GPU partitions) and Discoverer (CPU partition), and we are awaiting contact from MareNostrum for access to both its CPU and GPU partitions.

3.4 GPU porting and Kernel Optimisations

At the moment of this writing, five physics modules have been correctly ported to GPU using Cartesian coordinates: equations of compressible Hydrodynamics (HD), Magnetohydrodynamics (MHD), Special Relativistic Hydrodynamics (SRHD), Special Relativistic MHD (SRMHD) under ideal condition, that is, infinite conductivity and Resistive Relativistic MHD (ResRMHD). Notice that while SRMHD and ResRMHD partially overlap, they count a different number of conservation laws to be solved (the latter evolves the electric field while the former does not) and cannot be unified. Likewise, we also successfully implemented and ported to GPU: i) all Runge-Kutta time stepping schemes (from 1st to 4th-order), ii) finite-volume reconstruction methods (e.g., linear, Weighted Essentially Non Oscillatory (WENO), Parabolic, Monotonicity Preserving (MP) and iii) most ($\approx 80\%$) of the available Riemann solvers. For a given physics module, time stepping, reconstruction and Riemann solvers can be combined quite arbitrarily, depending on the user's requirements (e.g. more diffusive solvers will likely be more robust in handling problems with sharp gradients).

At first approximation, the computational intensive part of the code are all performed by the GPUs while the CPUs manage the overall initialization process and handle I/O operations. Kernels have been restructured by reorganising previous one-dimensional functions (called several times by sweeping direction) into fully three-dimensional kernels performing the actual computation in one single call per direction. Complex loops with extensive computations have often been split into two or more parts to facilitate GPU porting, rather than for performance optimisation. Additionally, the structure of the main arrays has been modified to promote coalesced memory access on accelerated kernels: Boundary (K1), Reconstruct (K2), Riemann Solver (K3), Right Hand Side (K4), Constrained Transport Update (K5) (described below and in Fig. 3).

Optimisation actions on these Kernels are discussed below while additional information may be also be found in the Deliverable D1.e1 (Review EIC Actions).

3.4.1 Boundary

The `Boundary()` function handles nearly all inter-process communication by setting both internal (that is, inter-processor) and physical boundary conditions on all of the sides of the computational domain, by filling ghost zones for both cell-centered and face-centered data arrays. This routine is fundamental as it handles nearly all inter-process communications and is invoked one time per Runge–Kutta Stage (e.g., 3 times for a 3rd-order time stepping.

The type of boundary condition at the leftmost or rightmost side of a given grid is specified by the integers `grid[dir].lbound` or `grid[dir].rbound`, respectively. If this value is non-zero, it indicates that the local processor borders a physical boundary. If the value is zero (indicating an internal boundary), two neighbouring processors sharing the same side fill ghost zones by exchanging data values. This process is repeated for each dimension and applies to both cell-centered and staggered data arrays.

Currently, our communication routines have been tested in the synchronous (blocking) implementation using MPI and the NVIDIA Collective Communications Library (NCCL). Fig. 4 reveals the current performance of the code on a weak scaling test up to 256 nodes (1 node = 4 GPUs or 32 cores when running on CPUs) on Leonardo.

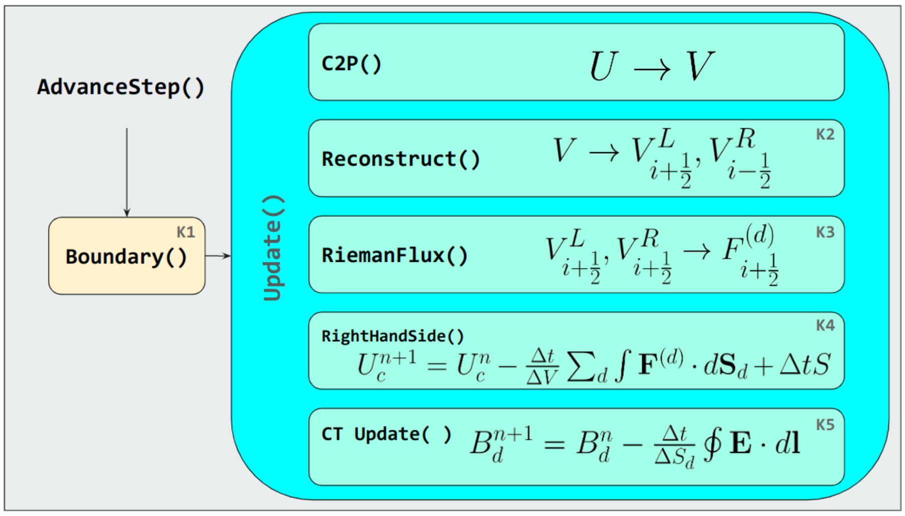

AdvanceStep()
Boundary()
K1
Update()
C2P()
$U \rightarrow V$
Reconstruct()
$V \rightarrow V^L_{i+\frac{1}{2}}, V^R_{i-\frac{1}{2}}$
K2
RiemanFlux()
$V^L_{i+\frac{1}{2}}, V^R_{i+\frac{1}{2}} \rightarrow F^{(d)}_{i+\frac{1}{2}}$
K3
RightHandSide()
$U^{n+1}_c = U^n_c - \frac{\Delta t}{\Delta V}\sum_d \int \mathbf{F}^{(d)} \cdot d\mathbf{S}_d + \Delta t S$
K4
CT Update( )
$B^{n+1}_d = B^n_d - \frac{\Delta t}{\Delta S_d} \oint \mathbf{E} \cdot d\mathbf{l}$
K5

Figure 3: Diagram of the Reconstruct-solve-average (RSA) strategy. The kernels are: Boundary (K1), Reconstruct (K2), RiemanFlux (K3), RightHandSide (K4) and CT_Update (K5)

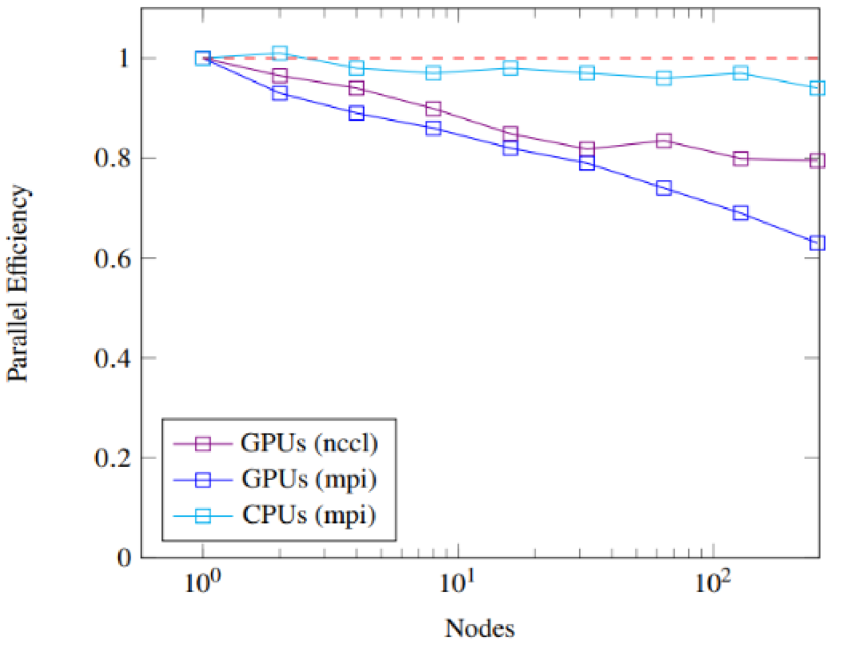


Figure 4: Weak scaling test on Leonardo running the 3D MHD Orszag-Tang problem. Here 1 node equals 4 GPUs (for GPU runs) or 32 cores (for CPU runs).

We are currently exploring the use of non-blocking MPI calls and the "async" clause in kernels to exploit concurrency, aiming for improved performance and scalability.

3.4.2 Reconstruct

The `Reconstruct()` function calculates the left / right primitive states at zone interfaces using one among several spatial reconstruction methods and it determines the scheme spatial accuracy and it has been ported to GPU using standard `#pragma acc parallel loop` directives from the OpenACC programming model. At each RK stage, this function is executed once per dimension (therefore 3 times for a 3D problem using a third-order RK method) using a user-defined reconstruction method, selected at compilation time. The reconstruction is typically based on piecewise polynomial interpolation subject to monotonicity constraints in order to avoid the Gibbs phenomena in the presence of discontinuities or steep gradients. At the time of this writing we have successfully ported: Linear, Parabolic, WENO reconstructions of 3rd and 5th-order and the 5th-order MP scheme.

In alternative to primitive variables, characteristic variables (i.e., obtained by projecting primitive variables onto the eigenvectors of the underlying equations) may also be employed at the cost of substantially increasing the operation count. These methods are essentially one-dimensional and employ information from adjacent zones. While the code offers different reconstruction methods- such as, Linear, Piecewise Parabolic, WENO, MP algorithm, we select linear (and later on fifth-order WENO and MP reconstruction) for optimisation (this is one of the most frequently used options).

In this routine, as in many others, the strategy has been to divide computations into several simple and efficient loops. This function consists of two separate kernels that operates on the whole active domain. These loops are designed to maximise the chances of coalesced memory access, by ordering the loop indices so that the inner loop index matches the fastest-changing index of the array within the kernel, whenever possible.

3.4.3 Riemann Solver

The `RiemannFlux()` function represents the most complex single kernel and computationally costly section of the code. This function computes the flux function at a zone interface given the input L/R states previously obtained during the reconstruction kernel. At the time of this writing, several Riemann solvers have been ported including: the Roe solver for 3/5 modules; the Harten-Lax-van Leer (HLL) for 5 / 5 modules, the Lax-Friedriechs solver (for 5 modules) the Harten-Lax-van Leer contact (HLLC) Riemann solver (for 4/5 modules); the Harten-Lax-van Leer discontinuity (HLLD) Riemann solver (for 2/5 modules) and the GFORCE solver (for 4/5 modules). Other solver includes the two-shock Riemann solver (for the HD module) and the Maxwell-Harten-Lax-van Leer contact (MHLLC) Riemann solver for the ResRMHD equations. This routine comprises a single kernel that could not be split into separate parts and involves numerous arrays and variables, resulting in substantial data movement between different levels of GPU memory affecting performance badly. The employment of C++ templated functions ensure that all variables are known at compile time, allowing them to reside in registers rather than main memory, thus optimising performance. Note that this routine also collects information necessary to the constrained transport update.

As for the previous kernel, acceleration has been achieved by means of standard `#pragma acc parallel loop` directives from the OpenACC programming model.

3.4.4 Right Hand Side

The `RightHandSide()` function evaluates the right hand side contribution for the conservative variable update and it is central to the computation process. It is invoked three times (once per direction) for each of the Runge-Kutta stages.

As for `Reconstruct()`, this function consist of several simple loops and it has been accelerated through standard OpenACC pragma directives. These loops are designed to maximise the chances of coalesced memory access, by ordering the loop indices so that the inner loop index matches the fastest-changing index of the array within the kernel, whenever possible. In the details, there are arrays that are overwritten for every direction so that the fastest index always matches the direction itself but there are also arrays whose element ordering is fixed (for example the main primitive and conservative variables arrays).

These 3D arrays contribute to the significantly different execution times for the routine across the three directions, which is a primary concern.

3.4.5 Constrained transport update

The `CT_Update()` function constructs the right hand side operator for the staggered magnetic (and electric when need) field components, in analogy with its cell-centered version (`RightHandSide()` function). Since these components have different spatial location and domain of existence, the kernel has been optimised using three accelerated loops via `#pragma acc parallel loop` compiler directives. This kernel is intrinsically multidimensional and cannot be reduced to a sequence of 1D operators. The function includes one kernel per direction (3 therefore), along with an additional kernels that operate exclusively for the ResRMHD module, ensuring control over the divergence of the electric field. Before the staggered update, information collected during the Riemann solver calls will be brought together to evaluate stable and properly upwinded electric field components at cell edges.

3.5 Alpha code release: Final remarks

With over than 50% of the physics modules ported from PLUTO to the HPC version, the new PLUTO is able to run and arriving at good performances (Fig. 4) on NVIDIA GPUs HPC cluster. The alpha version of the code is available at the previously indicated url, `https://gitlab.com/PLUTO-code/gPLUTO` and can as well be accessed from the original PLUTO web site (`https://plutocode.ph.unito.it`). gPLUTO will be distributed under the BSD 3-Clause License; users will be allowed to download directly gPLUTO from the GIT repository and share comments/suggestions/feedback through the PLUTO user forum. However, external users upload to local repository will not be allowed.

We finally point out that code porting is not over and several parts of it require further testing especially on HPC cluster. The features directly related to the use of HPC resources that we aim to achieve by the end of the project include a better scalability on MPI and NCCL and the implementation of OpenMP `#pragma` directives to allow the use of the AMD GPUs cluster.

4 BHAC

4.1 Introduction

BHAC is a multidimensional General Relativistic Magnetohydrodynamics (GRMHD) code that is mainly used to study accretion flows onto compact objects. BHAC has been designed to solve the GRMHD equations in arbitrary (stationary) space-times/coordinates and exploits AMR techniques with an oct-tree block-based approach provided by the MPI-AMRVAC framework (`https://github.com/amrvac/amrvac`). Originally designed to study Black Hole (BH) accretion in ideal GRMHD, BHAC has been extended to incorporate nuclear equations of state, neutrino leakage, charged and purely geodetic test particles, and non-black hole fully numerical metrics. BHAC has been employed in a number of studies of accretion into supermassive black holes and other compact objects. In addition, BHAC's results, after a General-Relativistic Ray-Tracing (GRRT) post-processing, can be used to compute synthetic observable images of BH shadows and the surrounding accretion flows. These calculations are performed with the GRRT Black Hole Observations in Stationary Spacetimes (BHOSS) code. The GRMHD simulation data produced by BHAC are used as an input for BHOSS to produce accretion flow and BH shadow images. A high-level description of the code and the main algorithms can be found in §4 of the deliverable D1.2.

4.2 CI/CD implementation

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. As mentioned above, this is the repository where all development related to SPACE CoE takes place. After each commit, a series of tests are performed, and only when all these tests are successfully passed, the commit is 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 is based on SRMHD and the second one incorporates GRMHD:

1. The former is the 1D shocktube benchmark test, which is passed when i) BHAC is compiled and the executable specific to the shocktube application is produced, and ii) a stable and non-crashing simulation runs for several iterations.

2. The latter is the magnetised spherical accretion onto a Schwarzschild BH, a static solution of GRMHD, that is passed when i) BHAC is compiled and the executable specific to the magnetised spherical accretion application is produced, ii) a stable and noncrashing simulation runs for several iterations, and iii) the L_2-norm of the difference between the profile of the rest-mass density at the final time and $t = 0$ (which is the exact solution) is less than a small number ϵ, where $\epsilon = 10^{-6}$.

BHAC is publicly available at the GitLab repository (`https://gitlab.itp.uni-frankfurt.de/BHAC-release/bhac`). Detailed documentation and directions on how to use the code are available at `https://bhac.science`. The development within the SPACE CoE happens on different branches of the GitLab repository (`https://code.it4i.cz/space_coe/bhac_mini_app`) hosted by IT4I, where CI/CD has been also implemented.

4.3 Running on EuroHPC JU clusters

To date, BHAC has been compiled and run on the CPU partitions of the EuroHPC JU clusters Karolina, Leonardo DCGP, LUMI-C, and Vega. The benchmark problems (1D shocktube and 2D magnetised spherical accretion onto a Schwarzschild BH) that were run on these machines gave as expected similar results. Also, the strong (see Fig. 5) and weak (see Fig. 6) scaling performance of BHAC on Leonardo DCGP and LUMI-C shows similar behaviour up to roughly 7000 and 5000 cores, respectively. Strong scaling tests on Karolina can be found in deliverable D2.1. Compilation on the rest of the EuroHPC JU clusters (MeluXina, MareNostrum and Discoverer) is currently ongoing.

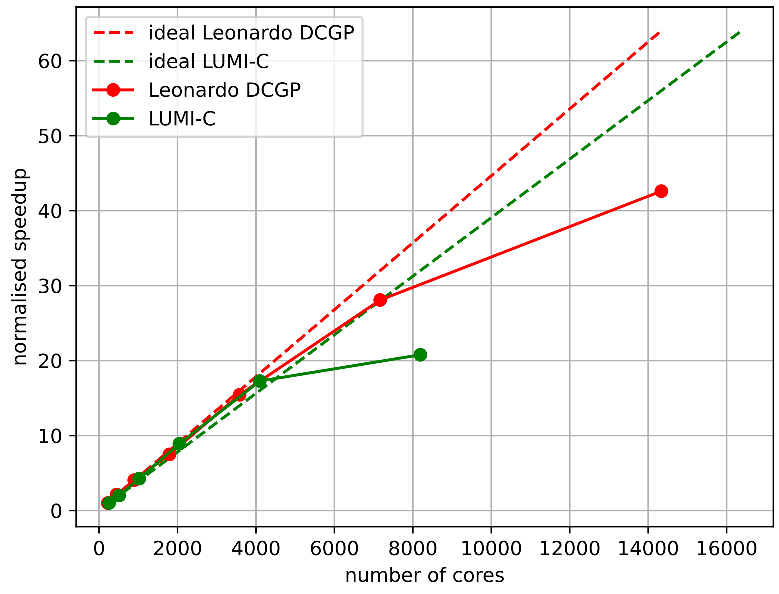


Figure 5: Strong scaling of BHAC on Leonardo DataCentric and LUMI-C.

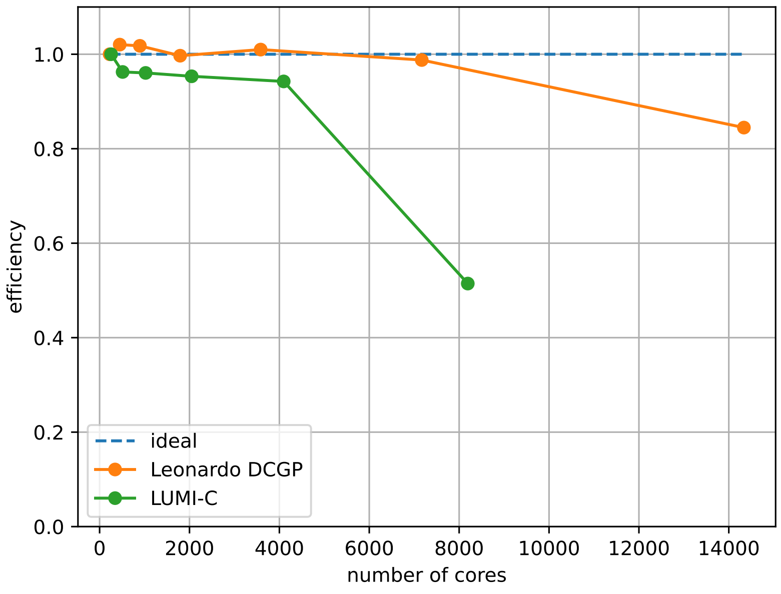


Figure 6: Weak scaling of BHAC on Leonardo DataCentric and LUMI-C.

Strong and weak scaling tests were performed on the EuroHPC JU supercomputers Leonardo DCGP @ CINECA and LUMI-C. Equipped with 1536 nodes with 112 cores each and 2048 nodes with 128 cores each, respectively, Leonardo DCGP and LUMI-C are ideal machines for scaling up to tens of thousands of cores. For the scaling the magnetised torus around a Kerr black hole described in D1.1 was used as a test case. The resolution used for the tests is $(r, \theta, \phi) = (1024, 448, 1024)$ on Leonardo DCGP and $(r, \theta, \phi) = (1024, 512, 1024)$ on LUMI-C resulting in a computational domain consisting of half a billion cells. The block based grid structure of BHAC divides the computational domain to a specific number of blocks containing a given number of cells.

BHAC requires that the number of blocks in a simulation is equal or greater than the number of cores used. Here we allocate one block to each core used, increasing gradually the number of blocks covering the computational domain. For the strong scaling this is done by decreasing the number of cells per block while keeping the total number of cells constant and for the weak scaling by keeping the number of cells per block constant while increasing the total number of cells. Strong scaling results are depicted in Fig. 5, BHAC scales good up to roughly 4000 cores and then because of issues related to load imbalance parallel efficiency drops. The weak scaling results are depicted in Fig. 6, the efficiency is above 98% up to roughly 7000 cores on Leonard DCGP and above 94% up to roughly 4000 cores on LUMI-C.

4.4 GPU porting and Kernel Optimisations

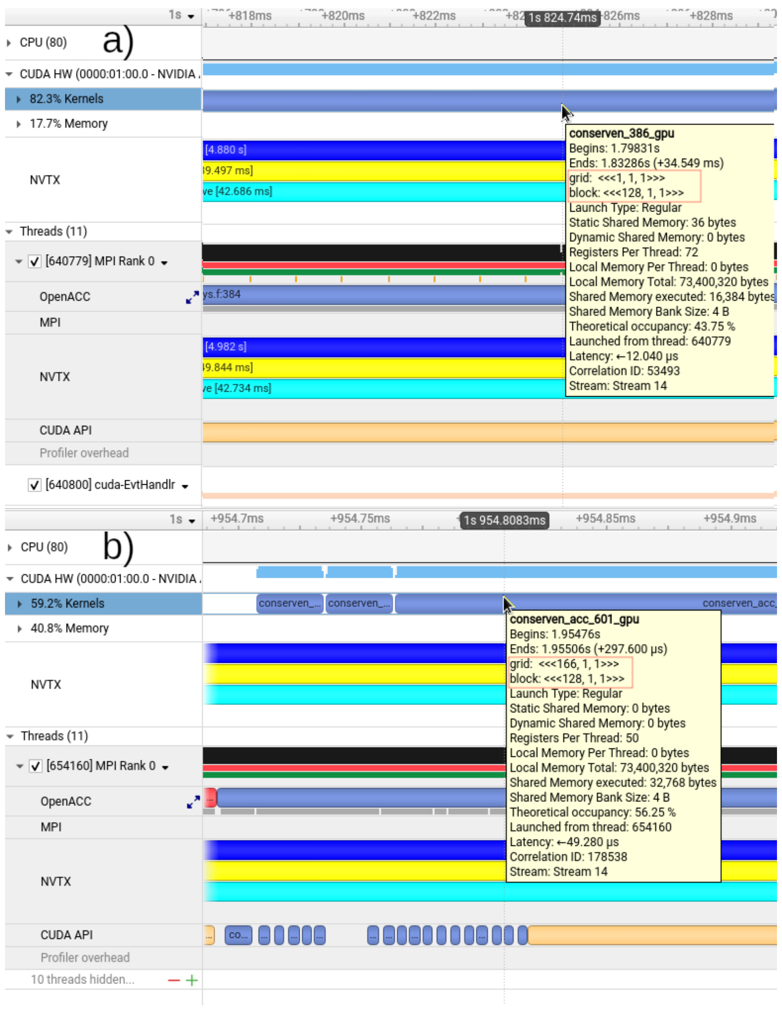
Currently, two modules have been initially ported to GPU: the primitive reconstruction and Riemann solver. Of course they need further testing and optimisation in order to be fully ported and to be able to function on a single as well as multiple GPUs. Notice that all development happens on a uniform grid for the reason that the MPI-AMRVAC framework that provides the AMR functionality to BHAC has not been ported to GPU—a task that is outside of the scope of SPACE CoE.

4.4.1 Primitive reconstruction

The module responsible for the conservative to primitive inversion has been ported to GPU using OpenACC. Only one of the three inversion strategies available in BHAC has been currently ported to GPU: the so-called *Entropy inversion method*, which is ideal for treating highly magnetised regions. The porting of the other two methods, i.e. 1DW and 2DW, will follow. Subsequently, the average or expected thread divergence due to the iterative nature of the Newton-Raphson method should be measured and in case divergence is drastically hindering performance, we should investigate methods to mitigate and improve performance, i.e. by CPU-side prepossessing.

4.4.2 Riemann solver

The Rusanov flux, also known as the Total Variation Diminishing Lax-Friedrichs scheme (TVDLF), has been ported to GPU using OpenACC. Next, the ported TVDLF solver will be optimized in order to increase the parallelisation efficiency and reduce, if not eliminate, the CPU-GPU data transfer. The sources of inefficiency are discovered by examining GPU traces obtained using NVIDIA Nsight Systems. As the code is in early stages of porting, we mostly focus on progressively eliminating data transfers by creating device versions of global and local data using OpenACC data directives. Also we focus on optimising initial ports of kernels which for various reasons are not fully parallelised (running sequentially or in a single block fashion). This is usually caused by compilers' inability to rule out possible data race or falsely reporting one. Example of detection of such an issue and performance gain resulting form its subsequent solving can be seen in Fig. 7. The optimisation of such problem usually take a form of some code transformation and modification of OpenACC directive. In this particular case even for small development dataset, the optimisation resulted in approx 116x speedup. As of writing of this document, not all such issues were solved and therefore we cannot present any performance results as they would be heavily skewed. The successful porting of the TVDLF solver will serve as a blueprint for the porting of additional solvers like the HLL, HLLC, etc, available in BHAC's infrastructure.



conserven_386_gpu
Begins: 1.79831s
Ends: 1.83286s (+34.549 ms)
grid: <<<1, 1, 1>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 36 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 72
Local Memory Per Thread: 0 bytes
Local Memory Total: 73,400,320 bytes
Shared Memory executed: 16,384 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 43.75 %
Launched from thread: 640779
Latency: ←12.040 μs
Correlation ID: 53493
Stream: Stream 14

conserven_acc_601_gpu
Begins: 1.95476s
Ends: 1.95506s (+297.600 μs)
grid: <<<166, 1, 1>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 50
Local Memory Per Thread: 0 bytes
Local Memory Total: 73,400,320 bytes
Shared Memory executed: 32,768 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 56.25 %
Launched from thread: 654160
Latency: ←49.280 μs
Correlation ID: 178538
Stream: Stream 14

Figure 7: Example of Nsight Systems traces of a) performance hindering semi-sequential kernel and b) fully parallel kernel. The problematic kernel can be easily identified by abnormally long runtime and subsequently verified by looking at launch configuration. In this case, the performance increase is approx. 116-fold.

For both modules, we will also focus on optimizing data structures for better data locality on CPU (vectorization) and GPU and optimize access pattern (loop order) for both CPU and GPU. In order to investigate the scalability of the GPU ported modules, we should move to multiple GPUs. While intra-node communication can be solved by direct GPU-to-GPU transfers, this is not always possible of inter-node communication. For early stages of GPU porting, the inter-node communication will be handled using MPI. Therefore, the great deal of attention has to be paid to data movement between CPU and GPU. The block structure of BHAC's numerical grid with overlapping ghost zones for communication will allow us to implement some communication-computation overlap to hide the latency of GPU-CPU data migration for inter-node communication. At a more advanced stage of the project we will try to use CUDA-aware MPI with GPU-CPU-NIC (Network Interface Controller) data transfers to achieve higher multi-GPU processing efficiency.

5 ChaNGa

5.1 Introduction

ChaNGa [3, 4, 1] is an N-body 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 [5] and pkdgrav [6] codes, the unique feature of ChaNGa is its incorporation of the Charm++ framework which enables highly efficient parallel scaling. To achieve this, Charm++ employs overdecomposition, that is to divide the work into many more work pieces than the number of available processors and let the Charm++ runtime system load balance by appropriately assigning pieces to real processors. During runtime, Charm++ applies dynamic load re-balancing strategies, to determine which work pieces should be migrated to new processors for better load balance. As the central feature of ChaNGa is it's tree-based gravity solver, the work pieces handled by the Charm++ framwork are vertical slices of the global tree, also known as tree pieces. The number of tree pieces is directly correlated to the given overdecomposition chosen by the user. 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 optimisation can be found in §8 of Deliverable 1.2.

During this initial development period we have setup git repositories, started to implement a more rigorous CI/CD, updated our general GPU implementation, and implemented a tree piece replication method to tackle the scaling issues found during the performance analysis (D2.1). The details of these improvements are outlined in the sections below.

5.2 CI/CD implementation

ChaNGa is publicly available on github (`https://github.com/N-BodyShop/changa`) and have been for sometime. 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 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). MHD which have before been only accessible through private branch is currently under a pull request to be reviewed and pushed into the public repository. The private MHD branch is still in use in the development of a more rigorous CI/CD as explained in the CI/CD section (which will eventually be pull requested and committed to the public repository). Both the public and private MHD branches have been mirrored to the IT4I GitLab (`https://code.it4i.cz/it4i-robertw/changa`) and the Höchstleistungsrechenzentrum Stuttgart (HLRS) GitLab (`https://codehub.hlrs.de/coes/space/changa/changa`), to provide a common access place for all the CoE codes. The public and private MHD branch are the development branches relevant to the SPACE CoE project and the improvements we aim to accomplish.

The public repository of ChaNGa already includes a basic 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`), and will be included as a submodule in ChaNGa for additional CI. 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.) are checked so that they reproduce the same result as previous versions of the code. The current pipeline uses public available GitHub runners.

5.3 Running on EuroHPC JU clusters

ChaNGa has, so far, been compiled and benchmarked on the Karolina-CPU, LUMI-C and Leonardo-DCGP clusters. In Fig. 8, we see the scaling of the single time step and the gravity module on Karolina, LUMI and Leonardo, for a 25 Mpc cosmological simulation with 2 billion particles at high redshift. From the figure we can see that there is loss of parallel efficiency at higher node counts for the single time step. This is because the runtime of gravity becomes comparable to that of domain decomposition (Domain Decomposition (DD)) and load balancing (Load Balancing (LB)). In a previous version of ChaNGa [1], an optimisation known as MetaLB was used to limit the overhead from DD and LB. This detects if there is an imbalance in the load of tree pieces and only performs a DD and LB step if this is the case. However, the MetaLB optimisation is currently not functioning properly in newest version, as such DD and LB is always performed at every step in the current version of ChaNGa. From the figure we can see that the scaling single time step differ between clusters, which comes down to different scaling in DD and LB. This is likely due to the difference in MPI and architecture between the clusters. From past testing we know that Open MPI (Karolina+Leonardo) struggles in handling the large number of messages that ChaNGa produces, therefore other MPI layers have been preferred (mostly MVAPICH2).

We have recently gotten access to MeluXina, VEGA, Discoverer and MareNostrum, and will begin porting and benchmarking on these systems in the upcoming months.

5.4 GPU porting and Kernel Optimisations

5.4.1 Remote communication (Gravity and SPMHD)

During our performance analysis (D2.1), it became clear that ChaNGa suffers from communication imbalance for highly clustered datasets involving all the main physics modules. We have tackled this issue 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. This preliminary version already shows promise for the merger case that we benchmarked in D2.1. This can be seen in Fig. 9, where tree piece replication significantly improves the scaling, both for the overall timestep and the gravity+SPH step. We have also found that using more communication threads than default can further improve the results. From the two plots we can see that gravity+SPH scales better than the overall timesteps and this is due to some modules (star formation + feedback) scaling less efficiently. The improvement of the two new optimisations can be seen in the projections traces for the 64 node case (Fig. 10), where we see a large reduction in idle time between the local and remote gravity calculation (colours and regions are same as in D2.1). Potentially with some more work and optimisation to the tree replication, we can further decrease this idle time. At lower CPU count, the tree replication code is slightly slower than the code without. We use an excessively high number of tree replications here (8), but this could likely be much lower. We plan to implement a parameter to choose the number of replicas. When the number of threads exceeds 8000, in this merger case, the problem size becomes a limiting factor due to the reduced number of tree pieces assigned to each thread. This reduction hinders load balancing, which often requires > 8 tree pieces per thread on average for good load balance. Although increasing the number of tree pieces might seem like a solution, it leads to smaller number of particles per tree piece, which increases overhead, which then becomes another limiting factor. We will prepare a larger merger case simulation and test scaling to higher node count.

There is still some cleanup and generalization that needs to be done for the tree-replication code. For example, right now the number of replicas is hardwired to be 8 and some functions need to be refactored and generalized. In addition, more testing should be done to figure out potential limitation or breaking points of the method.

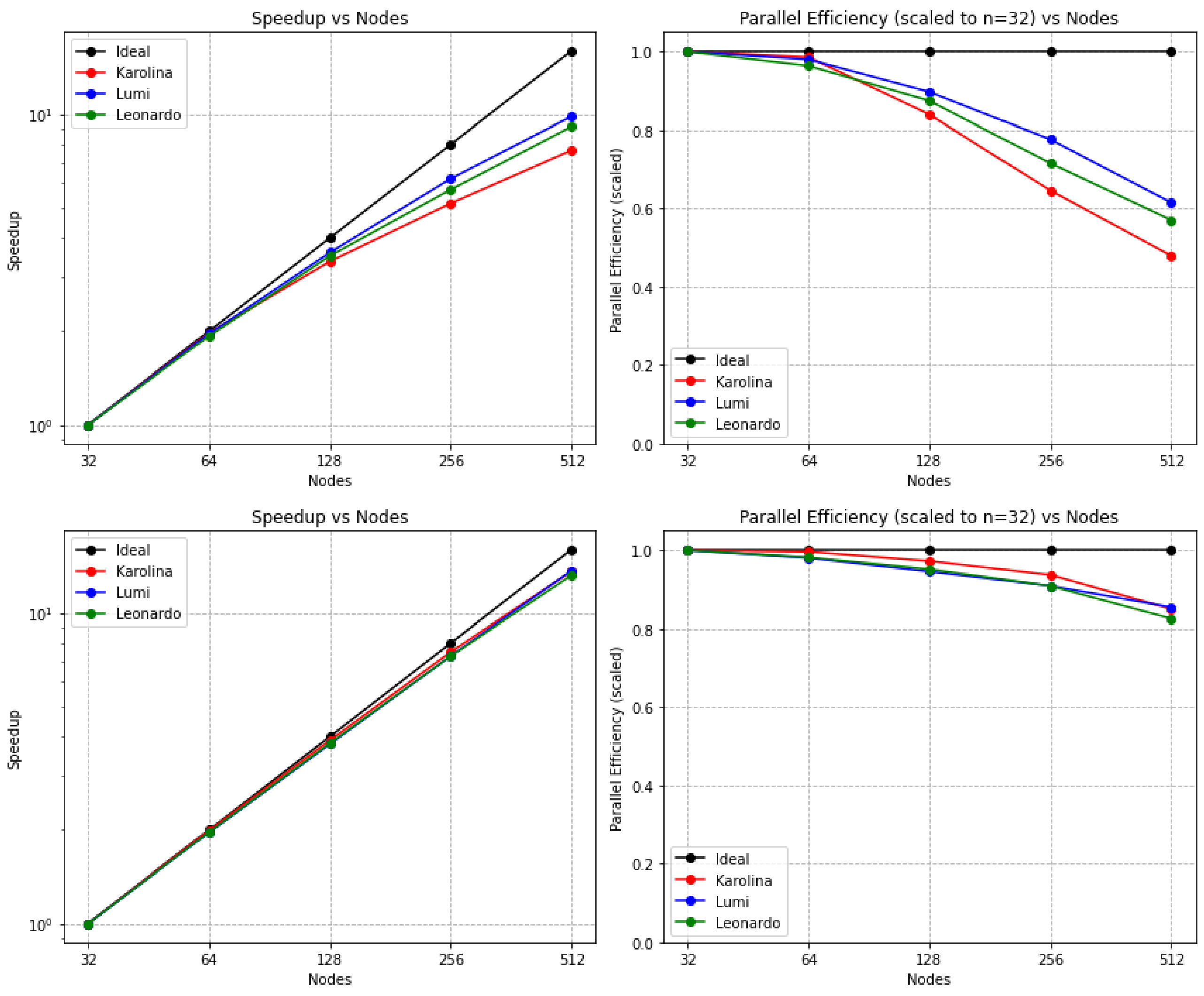


Figure 8: The top figure shows the scaling for a single time step, while the bottom show the scaling of only the gravity module. These simulations was performed on the Karolina-CPU, LUMI-C, and Leonardo-DCGP partitions using a cosmological box containing 2 billion particles within a 25 Mpc volume, using gravity only. The speedup curves have been normalized by the results from the 32-node run for each cluster. When the runtime of gravity becomes comparable to that of domain decomposition (DD) and load balancing (LB), the parallel efficiency declines. This is because DD and LB does not scale as well as gravity. The better single time step scaling seen in LUMI-C is due to better scaling of DD and LB (likely due to differences in MPI and architecture). We should note that this is without the use of the MetaLB optimisation used in [1](which limits the overhead by DD and LB), as this is currently not functioning properly in current version of ChaNGa/Charm.

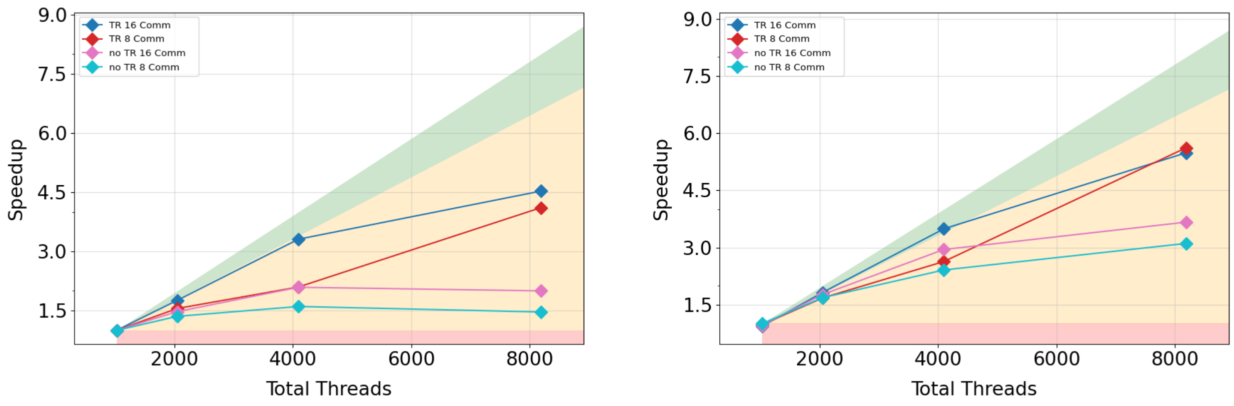


Figure 9: Scaling plots of the galaxy merger simulations. Left shows single time step scaling (can be compared with Deliverable D2.1 plot) and right show gravity+SPH module scaling. We can see much better scaling with the addition of tree piece replication and more communication threads.

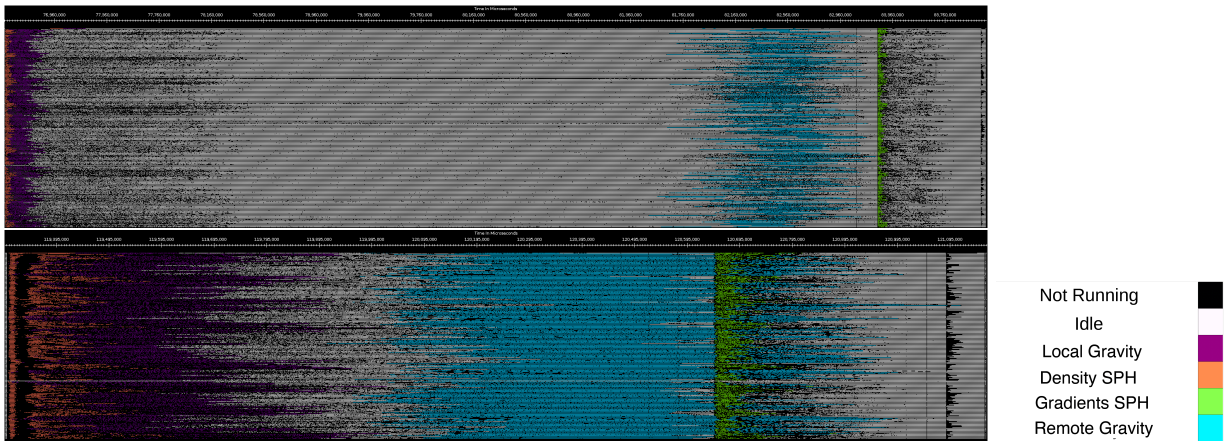


Figure 10: Projections traces of single time step of the merger simulation, where y axis shows all the active processors and x axis the time. The colors represent what each processor is working on or if it's idle (see legend to right). The top figure shows the result from the old version and the bottom shows the result from the version with tree replication and increased communication threads. We can see a massive reduction in the idle time between the local gravity calculation and the start of the remote gravity calculation. For more details about the regions see Deliverable D2.1

5.4.2 GPU related activities

The goal is to port several modules of the ChaNGa code to the GPU. The first modules which we will aim to port is the radiative cooling and SPMHD modules of the code. GPU porting is carried out in close collaboration with Spencer Wallace and Tom Quinn at the University of Washington, both co-developers of the ChaNGa code.

Before porting the cooling and SPMHD module, some work needed to be done on the general GPU implementation in ChaNGa. Previously, `workRequest` functions were used to allow ChaNGa to asynchronously interact with the GPU. These have become deprecated and do not function with the Charm++ performance tracing programme "Projections", so we have decided to remove these functions from the code. Instead, we will directly handle GPU requests in ChaNGa by submitting memory transfer/allocation requests, kernel launches, and callback assignments to CUDA streams. Each treepiece is assigned a CUDA stream in a round-robin fashion, and the number of available streams can now be controlled via a runtime parameter. This have all been implemented and pushed to the public git.

Additionally, while testing this new implementation in ChaNGa, we identified an issue during multi time stepping runs when only a few particles are on the smaller time steps. For these smaller timesteps, running on the GPU is slower than on the CPU due to the additional overhead. To address this, a runtime parameter has been added to switch to a tree walk on the CPU whenever the number of active particles falls below a certain threshold.

While both the gravity and cosmological boundary modules have been ported to GPU prior to the SPACE initiative, we want to continue to port the remaining computationally heavy modules (SPMHD, radiative cooling, feedback) to the GPU. Previously, we have noted that we would start porting the SPMHD module first, though we have decided to begin with the porting of the radiative cooling module instead. While the SPMHD module is in many situations the second most computationally heavy module, the radiative cooling can become the second most computationally heavy module in some conditions. This is because the radiative cooling is a semi-implicit module and that large integration times are needed when the cooling timestep is much smaller than the dynamical timestep. The radiative cooling is also easier to port to the GPU than the SPMHD. 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. Alternatively, 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).

6 FIL

6.1 Introduction

FIL, like BHAC, is a multidimensional GRMHD code. FIL is different from BHAC in the sense that it has been designed to solve the GRMHD equations in dynamical spacetimes. This allows FIL to simulate the maelstrom that are neutron star collisions. Whilst there is a greater computational cost for a full GRMHD code, there is a greater generality to the problems FIL can be applied to. FIL is a state of the art numerical relativity code; it uses fourth-order finite difference schemes to provide highly accurate simulations. FIL leverages the Einstein Toolkit (ET) which provides the infrastructure to enable numerical relativity simulations. FIL provides the computational modules, whereas the ET provides the AMR grid and memory management that FIL uses via the Carpet module. A new version of the Carpet module, CarpetX has recently been released, based on the AMReX library. CarpetX is compatible with CUDA and OpenACC and is designed for exascale computing. Therefore, to accelerate and improve FIL's scaling, a large part of our work will involve replacing Carpet with CarpetX. We can then focus on accelerating the computational modules in FIL using GPUs and optimising them. This is the general outline of the planned work. For further information on FIL, a high-level description of the code, and the main algorithm, please refer to §6 of the deliverable D1.2.

6.2 CI/CD implementation

The ET is publicly available, well documented, and well maintained (`https://github.com/EinsteinToolkit`). To operate FIL, the tutorials provided by the ET are sufficient. FIL is an upgraded version of the ET base fluid and spacetime computation modules. FIL is designed to be as similar as possible in user experience. FIL has to be made public by the end of 2025 and so work has been underway to improve the documentation and bring FIL inline with ET standards. The development within the SPACE CoE happens on different branches in the GitLab repository (`https://code.it4i.cz/space_coe/fil`) hosted by IT4I.

CI/CD, for FIL, has been implemented at the GitLab repository (`https://code.it4i.cz/space_coe/fil`) provided by IT4I and the corresponding runner is setup on IT4I's Karolina cluster. After each commit, a series of tests are performed to make sure the code functions as expected. The first test is a compilation test. Compiling FIL and the ET at the same time can take up to 30 minutes and so a pre-compiled version of the ET is stored on Karolina, and FIL is connected to it using symbolic links. When the runner pulls the modified version of FIL, it compiles FIL with the previously compiled ET. The second test is a simple print test to make sure the executable is functional. Finally, for the last test, FIL is used to solve the Tolman–Oppenheimer–Volkoff (TOV) equation, which is the equation for GRMHD model of a star. This is a robust and computationally challenging test and it requires the use of all the kernels that we have identified for optimisation. The numerical solution for TOV equation is validated with respect to an analytical solution. Due to numerical errors, there will be some oscillations around the analytical solution but this can be taken into account with a tolerance.

6.3 Running on EuroHPC JU clusters

To date, FIL has been compiled and run on the CPU partitions of the EuroHPC JU clusters Karolina and Leonardo. Compilation on the rest of the EuroHPC JU clusters is currently ongoing. The scaling tests for FIL are almost complete and will be presented in the next deliverable. FIL has also been compiled and run on the HLRS HAWK cluster and the SuperMUC cluster where we have recently performed scaling tests. In Fig. 11, we can see that FIL scales well up to 32000 cores for a unigrid setup. For the AMR scaling test, FIL preforms worse and scales poorly after 2560 cores.

6.4 GPU porting and Kernel Optimisations

We have planned to focus our acceleration efforts on BHAC until September. This will be done in collaboration with our partners at IT4I. The skills we will learn porting BHAC to GPUs using OpenACC can then be transferred to FIL. Once we have gained experience, we expect our work rate to increase significantly. BHAC was chosen to be the first code we work on as it is simpler to port as it being an isolated code whereas FIL is coupled strongly to the ET.

However, this does not mean that work on FIL will stop. We are, in the meantime, training our FORTH partners how to operate and understand FIL. The material that we produce in this training will be the bases for our training of the wider astrophysics community that we will produce as part of our training deliverables. Our

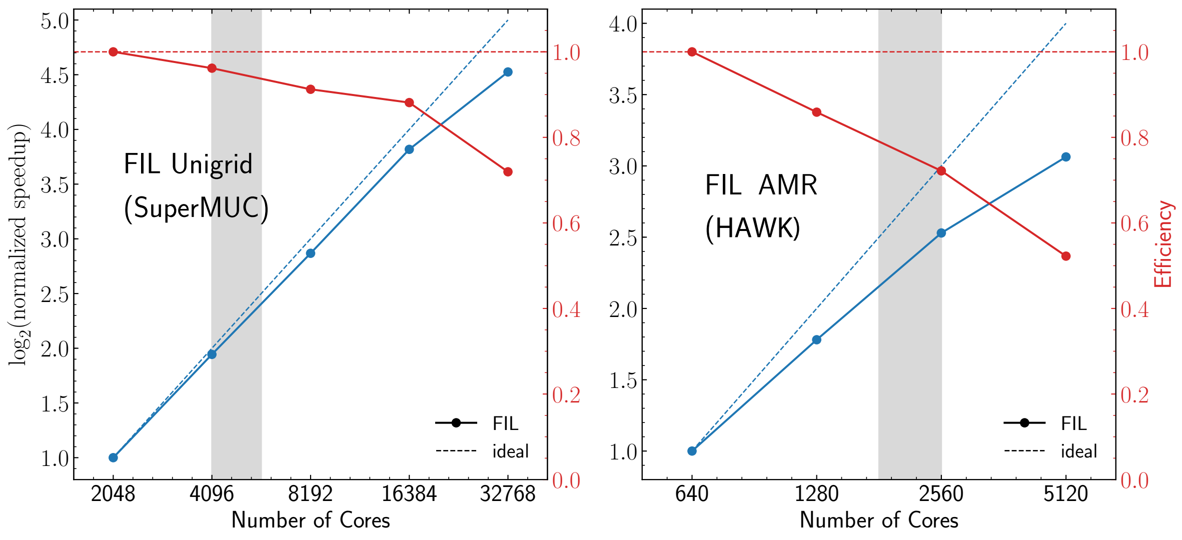


Figure 11: Scaling test of FIL AMR and Unigrid performance on HAWK https://www.cardiff.ac.uk/advanced-research-computing/about-us/our-supercomputers. In this picture dashed lines represent ideal scaling, solid lines are the current performances. Red lines are efficiency, blue lines normalised speedup

FORTH partners will profile and start the process of porting FIL to GPUs whilst the scientific partners work on BHAC. We will also be attending the ET conference in July. The focus of the conference will be learning what changes need to be made to codes to make them compatible with CarpetX (released in November 2023) which is the primary requirement for porting our code to GPUs. We are also planning on attending two NVIDIA Hackathons - one each for BHAC and FIL.

We have already collected and distributed all the reading material our FORTH partners need to conceptually understand FIL. We have set up good communication channels using the message hosting website FIL for seamless communication. We have created Jupyter notebooks on how to compile and run the tests that we will be using during our performance optimisations, namely head-on collision/TOV star simulations. We have sat with and worked with our FORTH partners whilst they go through the Jupyter notebooks as well as setting up bi-weekly training meetings.

6.4.1 Optimisation on Kernel 1: Carpet

The first kernel that we will start with is Carpet - we will work out how to make FIL compatible with CarpetX. As previously mentioned, CarpetX is based on the AMReX library and is therefore capable of working with CUDA and OpenACC as well as being optimised for exascale computing. Carpet must be replaced with CarpetX as having to transfer data constantly back and forth between GPU and CPU will limit any performance gains obtained using GPUs. Load balancing issues within Carpet were also identified as the main source of inefficacy within FIL and replacing Carpet should solve this issue. In other words, if we port FIL, the computational module, to GPU without switching to CarpetX, there will be constant data transfer between the processing units at every time step which will kill performance even on unigrid. We will gain a better understanding of the changes that need to be made to FIL to make it compatible with CarpetX in ET conference in July. We will be collaboratively work with the code developers of CarpetX in person.

CarpetX will act as the base of our mini app. Since FIL is coupled strongly to the ET, it requires a lot of the functionality that the ET provides. Therefore, we will start with CarpetX, work to make FILs modules compatible with CarpetX, and optimise them for exascale computing and accelerate them.

6.4.2 Optimisation on Kernel 2: Driver evaluate MHD RHS

During the profiling outlined in the deliverable D1.2, the driver evaluate MHD RHS kernel was found to have overall good OpenMP performance. Performance was throttled mainly by the bottlenecks in Carpet. To improve the OpenMP performance, we are going to use OpenMP region aggregation. As kernel 2 is a purely computational kernel, it has been identified as a good target for vectorisation and GPU offloading.

7 iPic3D

7.1 Introduction

Implicit Particle-in-cell 3D (`iPic3D`) [7] is a Particle-in-Cell (PIC) code developed to study plasma dynamics at the kinetic scale. In this open-source code (`https://github.com/CmPA/iPic3D`), 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 an 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*. A high-level description of these modules (and the code) can be found in Section 7 of Deliverable D1.2.

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-Friedrich-Lewy constraint. This allows us to choose time step sizes and grid sizes that can be $10 - 100$ larger than those used in traditional PIC codes.

The developments for `iPic3D`, within the framework of the `SPACE CoE`, are committed and pushed to `https://code.it4i.cz/space_coe/iPic3D`, hosted by IT4I and `https://github.com/CmPA/iPic3D`. Any changes to the IT4I GitLab repository is mirrored to `https://codehub.hlrs.de/coes/space/ipic3d/ipic3d`.

7.2 CI/CD implementation

The `SPACE_CPU` branch in `https://code.it4i.cz/space_coe/iPic3D` has CI/CD implemented. We consider a two-dimensional case of a thermal plasma, with a resolution of 112×56 and 20 particles per cell, which is evolved for 10 time cycles on 7×4 MPI tasks (cores). The input file corresponding to this test can be found in `iPic3D` $\longrightarrow$ `inputfiles` $\longrightarrow$ `CI_test.inp`. In order to avoid using excessive computational resources, we have opted for a relatively small-sized problem for the CI/CD tests. The results generated during this run is then tested against a reference data set, with the exact same simulation parameters, the data of which can be found in `iPic3D` $\longrightarrow$ `data` $\longrightarrow$ `ci_ref`. The bash scripts for these commands can be found in the `iPic3D` $\longrightarrow$ `scripts`.

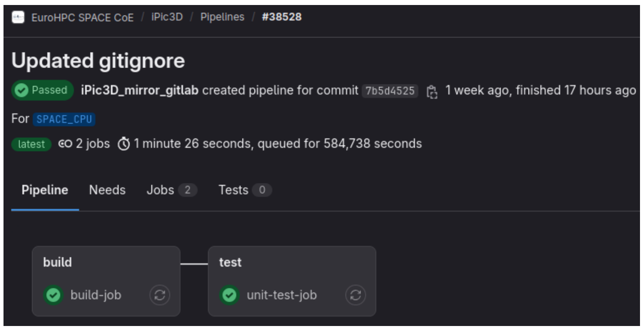
Any commits to the `SPACE_CPU` branch triggers a test of compilation, run, and comparison of resulting data with a reference data set (Fig. 12). We compare the difference in the electric and magnetic field across the two-dimensions: values of 0 in the differences of these quantities indicate that the latest commit yields the exact same results as that of the reference dataset (see lines 603 - 608 in Fig. 12). If these values are significantly greater than the machine precision, this would indicate incorrect results. As we are computing the error incurred in the various physical quantities individually, it would be straightforward to check for any mismatch of values corresponding to any of the physical parameters. The three modules in `iPic3D`, particle mover, moment gatherer, and field solver are inherently coupled, and as such, all three modules are tested for the CI test. Furthermore, the initialisation of a problem and writing of the resulting data set to HDF5 files are also implicitly tested. Overall, the entire fundamental code base is tested during a CI test and failure in any one of these modules would result in a failed test. We note that the CI pipeline can be run by the user on demand to test the recent developments at any point in time.

In the near future, we aim to automate strong and weak scaling tests along with the test for verification of correctness of the code to gain a better understanding of the code performance in terms of the speedup achieved. We are working on developing a similar CI test for the `GPU_OpenACC` branch. The finalised version of this will be implemented following the ongoing optimisations of the particle mover module.

7.3 Running on EuroHPC JU Clusters

We have successfully compiled and run `iPic3D` on Leonardo DCGP, Leonardo Booster (with the particle mover module running on the GPU), LUMI-C, and Karolina CPU. We have applied for EuroHPC JU Call for Proposals for Development Access to run `iPic3D` on MareNostrum5 GPP, MeluXina CPU, Discoverer, and VEGA CPU. In the next deliverable, we will present strong and weak scaling tests on these aforementioned clusters.

Additionally, `iPic3D` has been successfully compiled and ran on NVIDIA Grace (ARM Neoverse v2) with `nvhpc` compiler by Elisabetta Boella (E4) who is also directly involved in facilitating GPU offloading of `iPic3D`. She is currently running tests to compare the performance on ARM architecture with that of `x86_64`.



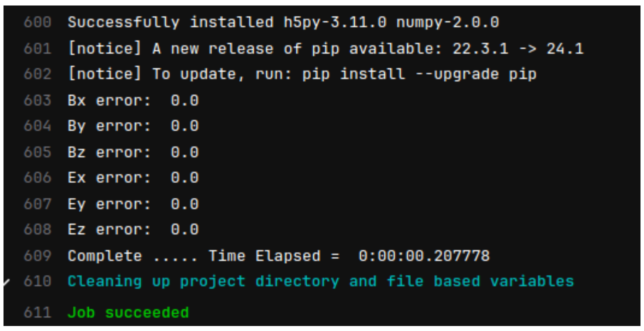
EuroHPC SPACE CoE / iPic3D / Pipelines / #38528

Updated gitignore

Passed iPic3D_mirror_gitlab created pipeline for commit 7b5d4525 1 week ago, finished 17 hours ago

For SPACE_CPU

latest 2 jobs 1 minute 26 seconds, queued for 584,738 seconds



```
Successfully installed h5py-3.11.0 numpy-2.0.0
[notice] A new release of pip available: 22.3.1 -> 24.1
[notice] To update, run: pip install --upgrade pip
Bx error:  0.0
By error:  0.0
Bz error:  0.0
Ex error:  0.0
Ey error:  0.0
Ez error:  0.0
Complete ..... Time Elapsed =  0:00:00.207778
Cleaning up project directory and file based variables
Job succeeded
```

Figure 12: **Left:** figure shows the basic test of the CI/CD pipeline that needs to be passed for the `SPACE_CPU` branch. The "unit-test-job" carries out tests for the particle mover, moment gatherer, and field solver, in addition to problem initialisation and writing data to files. **Right:** screenshot of the final stages of the CI test - one can see that the errors in the electric and magnetic fields, compared to the reference solution are exactly 0, thereby indicating correct results of the latest commit in `https://code.it4i.cz/space_coe/iPic3D`.

7.4 GPU porting and Kernel Optimisations

7.4.1 GPU porting: Particle Mover

We have offloaded the particle mover module to GPU using OpenACC and we are now working on optimising the performance of this module on GPU. The particle mover kernel contains `#pragma acc parallel loop` to iterate over all the particles. This is the first module, of `iPic3D`, to be ported to GPUs within the framework of SPACE CoE. We have simplified the electric and magnetic field arrays in a way that enables vectorisation. Vectorisation of the field arrays, as opposed to using multiple pointers, avoids potential memory leaks and as well as provides some speedup. The performance of the GPU code is being analysed with NVIDIA Nsight Systems (`https://developer.nvidia.com/nsight-systems`) to identify scope for optimisation, analyse the performance of the module under consideration, test for misaligned memory accesses, memory leaks, potential race conditions, and necessary synchronisation breaks (see Fig. 13). We have noted several instances of data movements to and from the host and the device and many page faults.

To test the performance of the GPU offloaded particle mover, we consider a simple test case where we discretise the two-dimensional computational domain with 640 and 320 grid points along the X and Y directions, respectively. We consider five different values for the number of particles per cell (with the rest of the parameters being exactly the same as the one for the CI test), and we report the runtimes for the particle mover, with and without GPU support, in Table 1. Let us clearly state that the runtimes for the particle mover, with GPU support, includes the time needed for data transfers to the GPU and back to the CPU (as the moment gatherer and field solver still run on the CPU). This is why we do not observe large speedups for the GPU offloaded part. Whilst it is essential that the data transfer to and from the device, i.e., GPU is minimised, our current implementation requires transfer of the relevant data to and from the device at every time step. This is due to the implicit nature of `iPic3D`: the results yielded by the particle mover (on the device) have to be transferred back to the host to compute the moments and the fields. The data transfer between the host and the device will be the last stage of optimisation once all three modules have been ported to GPU. This test case was run on a single node with 32 MPI tasks and with a single NVIDIA A100 GPU on Leonardo Booster.

Optimisation 1

As a first step towards optimising this kernel, we investigate asynchronous data transfer to hide the memory access latency performance enhancements using `cudaMemPrefetchAsync` (Listing 1). We asynchronously prefetch a range of managed memory on the device for the particle velocities and positions and the electric and magnetic fields that are stored on the grid. Prefetching helps to manage data movement explicitly, ensuring that memory is available on the right device at the time of usage without substantial overhead. In Table 1, we observe some statistically significant improvement in performance whilst prefetching the memory only for relatively large test cases (i.e., where the number of particles per cell are 80×80 and 90×90). We expect to obtain even larger performance gains when we move to multi-GPU systems. As transferring large amounts of data to multiple devices across different nodes can be time-consuming, we expect asynchronous prefetching to play a crucial

Particles per cell	**CPU** (Time elapsed in sec)	**GPU** (No Prefetching) (Time elapsed in sec)	**GPU** (Prefetching) (Time elapsed in sec)
20×20	28	21	22
40×40	112	86	84
60×60	252	190	190
80×80	441	338	331
90×90	567	429	420

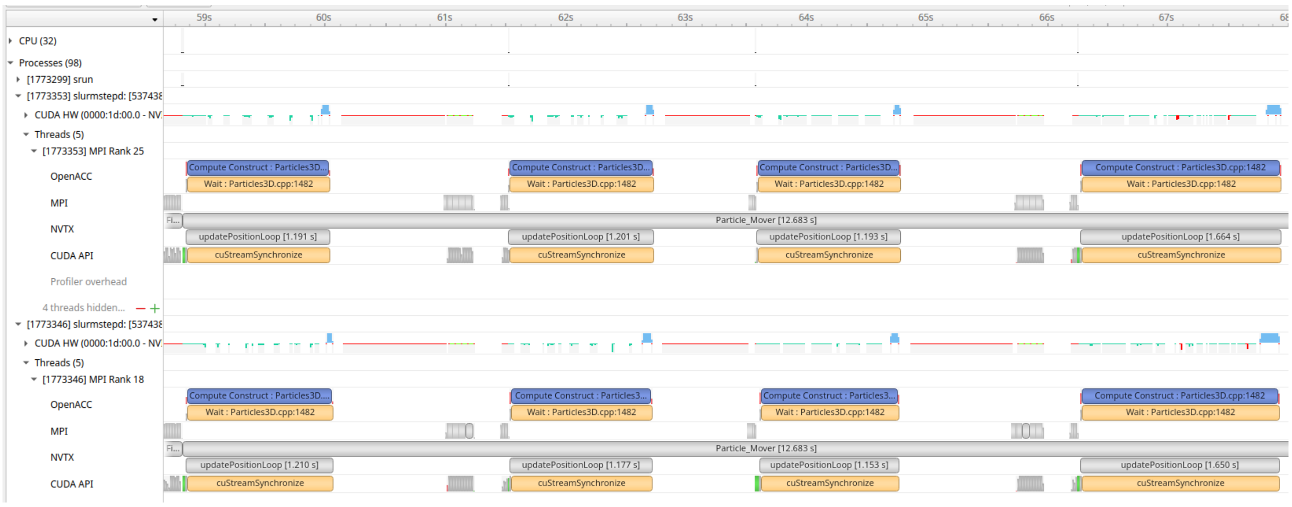
Table 1: Comparison of the computation cost, in terms of time elapsed in seconds, of the particle mover on 32 CPU cores (MPI parallelised) and on a single NVIDIA A100 GPU on Leonardo Booster for five different configurations of particles per cell. It is to be noted that the time elapsed reported for the GPU runs also takes into account the time needed for data transfer from the host to the device and vice versa. We expect significantly higher speedups once we finish offloading the other two modules and the entire data remains exclusively on the device for as long as possible.

role to achieve meaningful speedups. Code performance and scaling on multiple GPUs would be a subject of consideration in the upcoming deliverables.

Optimisation 2

Based on the recent POP3 analysis conducted by Radim Vavrik (IT4I), we have identified a potential scope for improvement in the particle mover module. The particle mover computes the velocities and positions of the particles at any given time step once the electromagnetic fields have been computed. It may so happen that not all particles end up in the correct MPI subdomain. In order to get the particles back to the correct subdomain, this communication is done only with six neighbouring subdomains along the three spatial dimensions. There is no communication between the corners. This requires local MPI communications, followed by an instance of `MPI_Barrier()`. It may so happen that not all particles end up in the correct MPI domain after just one communication (this is true for the particles that may have moved across the corners). If the particles happen to move across a corner, they may have to loop around the domain in multiple directions to get back to the correct domain. This necessitates multiple local MPI communications followed by multiple instances of `MPI_Barrier()`. Although this is fundamentally a simple approach to get the particle particles back to the correct subdomain, multiple loops over local communications and `MPI_Barrier()` can constitute a serious performance bottleneck in the particle mover module.

To remedy this, we will rewrite this communication part in such a way that each MPI subdomain communicates **only once** with all of its neighbouring 26 subdomains in 3D geometry. This can alleviate the need for looping through multiple instances of local communications and `MPI_Barrier()`. Let us note that this optimisation part applies to both the CPU and GPU versions of the code.



CPU (32)
Processes (98)
[1773299] srun
[1773353] slurmstepd: [537438
CUDA HW (0000:1d:00.0 - NV
Threads (5)
[1773353] MPI Rank 25
OpenACC
MPI
NVTX
CUDA API
Profiler overhead
4 threads hidden...
[1773346] slurmstepd: [537438
CUDA HW (0000:1d:00.0 - NV
Threads (5)
[1773346] MPI Rank 18
OpenACC
MPI
NVTX
CUDA API
59s
60s
61s
62s
63s
64s
65s
66s
67s
Compute Construct : Particles3D.cpp:1482
Wait : Particles3D.cpp:1482
Particle_Mover [12.683 s]
updatePositionLoop [1.191 s]
updatePositionLoop [1.201 s]
updatePositionLoop [1.193 s]
updatePositionLoop [1.664 s]
updatePositionLoop [1.210 s]
updatePositionLoop [1.177 s]
updatePositionLoop [1.153 s]
updatePositionLoop [1.650 s]
cuStreamSynchronize

Figure 13: Screenshot of the analysis of the particle mover module, being run on a single GPU using OpenACC, using NVIDIA Nsight Systems. We are profiling the code with `nvtx` to better understand the compute regions, data transfer, the wait times, and synchronisation.

```
#ifdef GPU_PREFETCHING
    cudaMemPrefetchAsync(u, sizeof(double)*nop, 0, 0);
    cudaMemPrefetchAsync(v, sizeof(double)*nop, 0, 0);
    cudaMemPrefetchAsync(w, sizeof(double)*nop, 0, 0);

    cudaMemPrefetchAsync(x, sizeof(double)*nop, 0, 0);
    cudaMemPrefetchAsync(y, sizeof(double)*nop, 0, 0);
    cudaMemPrefetchAsync(z, sizeof(double)*nop, 0, 0);

    cudaMemPrefetchAsync(Ex_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
    cudaMemPrefetchAsync(Ey_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
    cudaMemPrefetchAsync(Ez_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);

    cudaMemPrefetchAsync(Bx_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
    cudaMemPrefetchAsync(By_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
    cudaMemPrefetchAsync(Bz_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);

    cudaMemPrefetchAsync(Ex_ext_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
    cudaMemPrefetchAsync(Ey_ext_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
    cudaMemPrefetchAsync(Ez_ext_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);

    cudaMemPrefetchAsync(Bx_ext_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
    cudaMemPrefetchAsync(By_ext_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
    cudaMemPrefetchAsync(Bz_ext_d, sizeof(double)*(nxn*nyn*nzn), 0, 0);
#endif
```

Listing 1: Asynchronously prefetch the memory (requires `CUDA`) on the device for the particle velocities `(u, v, w)`, particle positions `(x, y, z)`, electric field `(Ex_d, Ey_d, Ez_d)`, magnetic field `(Bx_d, By_d, Bz_d)`, and the external electric and magnetic fields `(Ex_ext_d, Ey_ext_d, Ez_ext_d, Bx_ext_d, By_ext_d, Bz_ext_d)`.

7.4.2 GPU porting: Moment Gatherer

The moment gatherer module computes the different moments, namely, the charge density, the current density of each species, and the pressure tensor. Computing these quantities requires one to iterate over N_s particles of n_s species. This requires several local internode communications to send and receive boundary data. This becomes necessary as to compute the current and density at any given grid point, one needs to compute the effective contribution from all particles.

Currently, we are starting to work on offloading the moment gatherer module to GPU, which includes using NVIDIA Nsight Systems to identify scope of improvement and optimisation of the GPU code. For this module, we aim on obtaining around 40 − 50% of the memory bandwidth on the GPUs.

7.4.3 GPU porting: Field Solver

The field solver is the least expensive module of `iPic3D`, however, computing the electromagnetic fields demands a plethora of collective communications that can be an impediment to the code performance. We use the GMRes module of the publicly available `PETSc` library to compute the solution of the Maxwell's equations to obtain the electromagnetic fields.

We aim on using the `CUDA`-based GMRes module (of `PETSc`) for the field solver for NVIDIA GPUs on the Leonardo Booster partition and the `HIP`-based GMRes module for AMD GPUs on LUMI-G. Due to the multiple global communications (or synchronisations) needed, our aim is to achieve around 20 − 40% of the peak memory bandwidth on the GPUs. A relatively low throughput on the cheapest module should not severely deteriorate the overall code performance. In case it does, we will explore the possibility of using other publicly available libraries for the field solver, for instance the `Intel Math Kernel` library (`https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html`) or the NVIDIA `AmgX` library (`https://developer.nvidia.com/amgx`). We will also explore the possibility of using exponential integrators and polynomial interpolation methods [8, 9] that can be attractive for massively-parallel architectures, such as GPUs. We will decide on these alternatives if the GPU version of GMRes, which we will develop, fails to achieve significant speedup relative to the other two modules.

7.4.4 Research interaction with CINECA

We are in the final stages of optimisation of the particle mover module on the GPUs. In September, Pranab Deka (KU Leuven) will visit CINECA, Bologna for a two-week period to work in close collaboration with Nitin Shukla, Alessandro Romeo, and Elisbetta Boella (E4) to **finalise the porting of all three modules to GPUs**, which will run on Leonardo Booster. Following this visit, we will continue working on optimising the different modules in order to obtain high scalability on multiple GPUs on Leonardo. We will also start working on compiling, running, and benchmarking the so-developed GPU version of `iPic3D` on AMD GPUs on LUMI-G. The progress and output of which will form a significant part in the next deliverable (due December 2024).

8 RAMSES

8.1 Introduction

RAMSES [10, 11] is an AMR code that is used to study astrophysical fluid dynamics and the formation of structures in the Universe. It is based on an oct-tree structure, where parent cells 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. A high-level description of the code and its algorithms can be found in Section 8 of Deliverable D1.2.

8.2 CI/CD implementation

RAMSES is publicly available on Bitbucket (https://bitbucket.org/rteyssie/ramses/src/master/). This the main repository which is maintained using the version control system git. It also contains the stable releases which are delivered once a year, usually after the RAMSES User meeting. This repository is not aimed at being the one hosting the developments we conduct in the SPACE CoE. The development version is hosted on the local GitLab platform at CRAL, which is publicly available (https://git-cral.univ-lyon1.fr/hpc/space/ramses). This repository is mirrored to the GitLab account hosted by IT4I in order to setup a CI/CD pipeline targeted at parallel tests on an EuroHPC JU (https://code.it4i.cz/space_coe/ramses) cluster. Once validated, the developments will be merged to the main repository, following the policy that has been discussed during the last RAMSES User Meeting (pull request, review by RAMSES developers external from SPACE, and merge).

The CI/CD in place for the public version of RAMSES consists of an automatic test that is integrated in the CI pipeline. It consists of a build (make) of the code which is done after each commit. It is run on the Bitbucket CI/CD facilities. Then, an automatic test suite is run once a day on a dedicated computer hosted by the main developer Romain Teyssier. The results are then sent to the Bitbucket wiki to keep track of the daily results. The test-suite is thus not integrated to the CI pipeline and, as currently designed, the last commit is not automatically rejected if it fails. We participated at the last RAMSES User Meeting held in April 2024. The RAMSES community decided to move the public version from Bitbucket to GitHub, and at the same time improving the CI/CD workflow. We will participate to this effort, and the work done in the framework of SPACE CoE will thus benefit a larger community.

Within the framework of SPACE CoE, we aim to optimise the CI/CD pipeline and automate it. We have set up a CI pipeline to compile the code and run a short test suite on the CRAL GitLab. The test suite is done on physical setups (Sod tube, implosion, advection) that test the main kernel of RAMSES, and in particular, the hydro module which we have identified for optimisations. Currently, the CI pipeline is carried out after each commit. The development repository is now mirrored on the GitLab account hosted by IT4I (https://code.it4i.cz/space_coe/ramses). It is automatically updated after each commit on the development CRAL GitLab repository. The next step for the upcoming weeks is to setup an automatic conditional CI/CD pipeline that runs independently at CRAL and at IT4I.

This will enable us to maintain a compiling and working version on Karolina. We envision to deploy this strategy on all EuroHPC JU clusters (if GitLab servers are available) in order to maintain the version of RAMSES on the different EuroHPC JU clusters.

Our future plan for CI/CD for the coming months is as follows:

- optimise the CI pipeline to only compile the code after each commit and run the test suite only once a day on the CRAL development repository;
- setup the conditional pipeline to run the test suite after each commit at IT4I in order to test the parallel implementations (MPI and OpenMP);

8.3 Running on EuroHPC JU clusters

We have installed and benchmarked RAMSES on the EuroHPC JU clusters (CPU only) - Karolina, Leonardo DCGP, and LUMI-C. We ran the three test cases that we presented in D1.1. Fig. 14 shows a strong scaling test performed with RAMSES on the EuroHPC clusters Karolina, Leonardo DCGP, and LUMI-C.

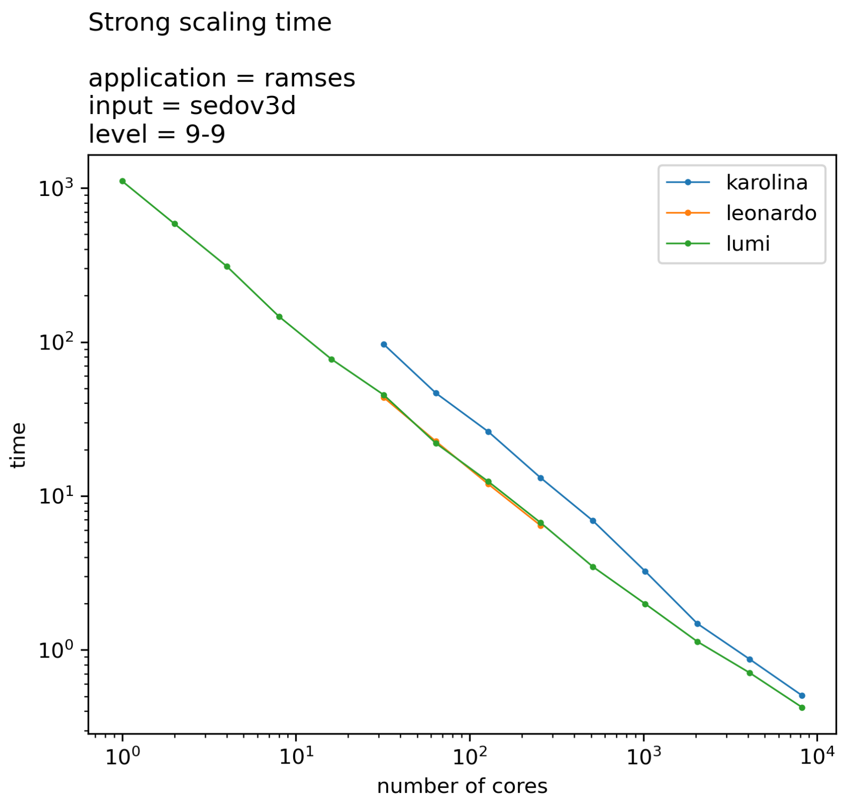


Figure 14: Strong scaling of RAMSES on a Sedov-Taylor blast test. The grid corresponds to the small configuration of RAMSES test case 1, i.e. a resolution of 512^3. The total wall-clock time is given as a function of the number of MPI processes. Overall, the Karolina, Leonardo DCGP, and LUMI-C clusters show similar performance.

8.4 GPU porting and Kernel Optimisations

In the first year, we have investigated the OpenMP porting of RAMSES on both CPU and GPU as a proof of concept. We report our results in the following sections.

8.4.1 CPU Optimisation on Kernel 1: Godunov solver

The Godunov solver is the most time consuming kernel for pure hydrodynamical simulations. It has been selected for performance analysis and optimisations in D1.2 and D2.1. From D2.1, the kernel already looks well-optimised on CPU architectures. The challenge is to port it on different hardware platforms (ARM, GPUs).

The computational part of the hydro kernel module of RAMSES mainly consists of assembling vectors (of a given size NVECTOR) and calling a Godunov solver onto them. Parallelising the loop over these vectors is a good solution to enable the use of multiple threads concurrently inside a subdomain.

With this solution, each thread assembles its own vector, consisting of a portion of the mesh and its neighbourhood. Then, each thread calls the same Godunov solving routine in order to numerically solve the hydro equations at the current time iteration. At the end, each thread writes back its results into the main arrays holding the physical variables of importance regarding the hydro module.

As a first step, we isolated the kernel and implemented OpenMP instructions.

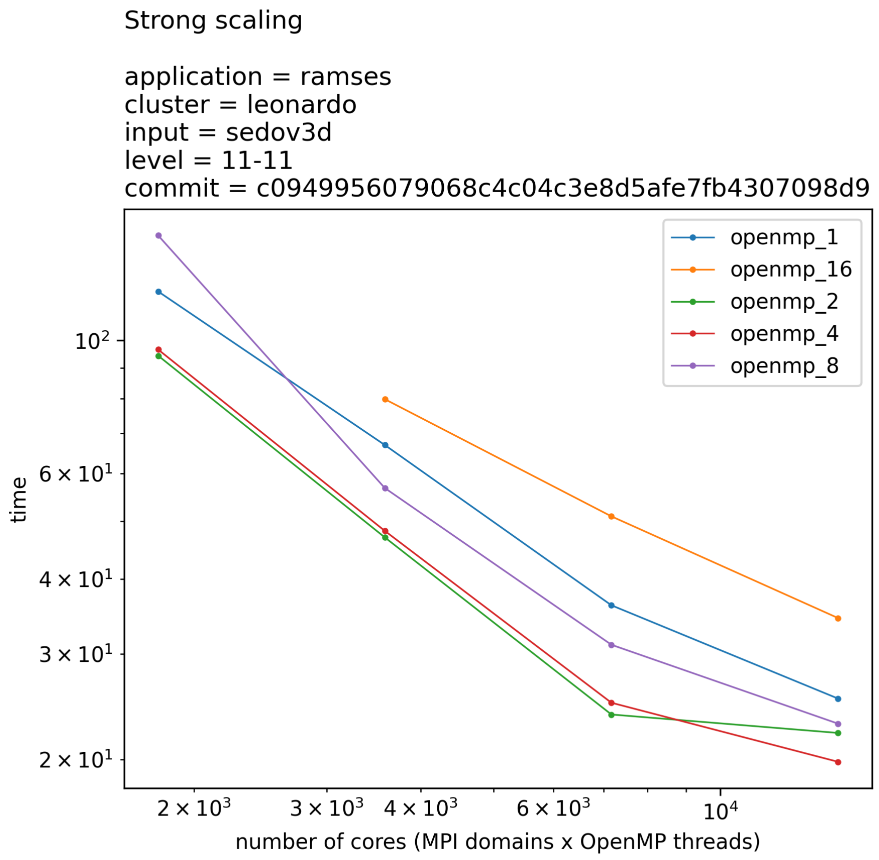


Figure 15: Strong scaling of RAMSES on a Sedov-Taylor blast test with different number of OpenMP threads with a fixed number of cores (MPI ranks × OpenMP threads)

```fortran
subroutine godunov_fine(ilevel)
    #ifdef OPENMP
      use omp_lib
    #endif

    ! Loop over active grids by vector sweeps
    ncache=active(ilevel)%ngrid
    !\$omp parallel do default(none) shared(ncache,ilevel,active)
   private(ngrid,ind_grid)
    do igrid=1,ncache,nvector
        ngrid=MIN(nvector,ncache-igrid+1)
        do i=1,ngrid
            ind_grid(i)=active(ilevel)%igrid(igrid+i-1)
        end do
        call godfine1(ind_grid,ngrid,ilevel)
    end do
end subroutine godunov_fine
```

It already shows a significant gain over the pure MPI approach in terms of strong scaling potential. Indeed, on the Sedov3D test case run on Leonardo DCGP, with a mesh size of 2048^3, the Fig. 15 below shows that, using the same number of 14336 cores by employing 4 OpenMP threads and 3584 MPI processes achieves better strong scaling with a gain of 20% over the full MPI version. This benchmark has been performed using the GNU Fortran compiler. We now need to check if comparable performance is achieved using the Intel Fortran compiler. The first OpenMP parallelization has been produced in the frame of the SPACE CoE.

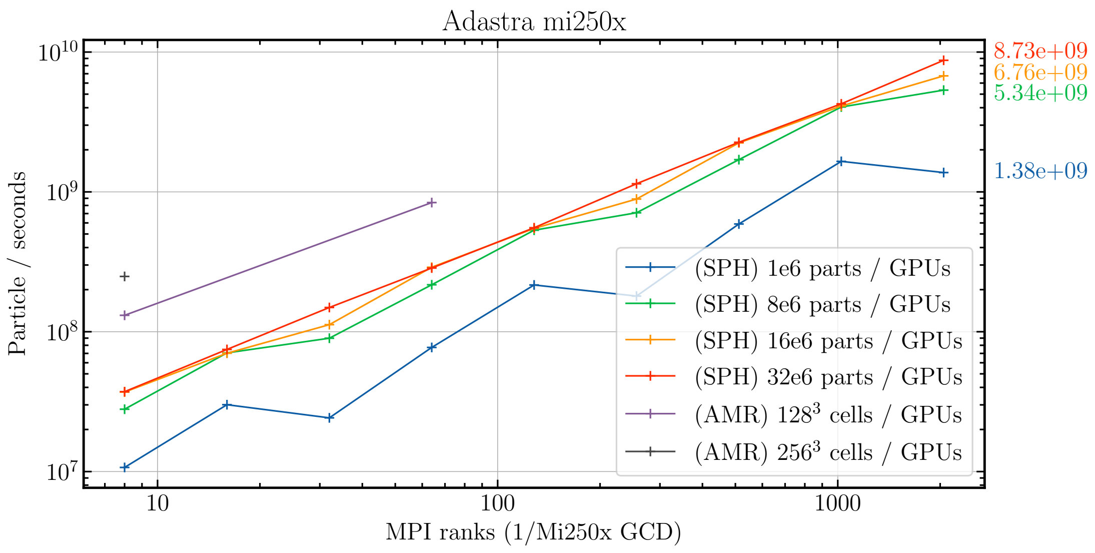


Figure 16: Weak scaling of SHAMROCK-SPH and SHAMROCK-RAMSES on Adastra that represents the total number of cells or particles updated per second on the entire computational domain for a pure hydro test. The first attempt to run a Godunov AMR solver (purple and grey) already shows better performance than the SPH kernels. Indeed, 50 neighbours are required for SPH, while the kernel is more compact for the grid and allows to update more cells per second.

8.4.2 GPU porting

Some pursuits to port RAMSES to GPUs have already been made, however, without success for the AMR part. The granularity of the main kernels is too small and too much time is spent in transferring data between CPUs and GPUs. This is a structural issue of RAMSES that makes it poorly compatible with GPUs and cannot be fixed without major architectural changes in the code. To illustrate this, we have ported the Kernel 1 (Godunov solver) on GPUs using OpenMP directives. The acceleration of the code executed using a single NVIDIA A3000 GPU is only a factor < 10 compared to a single CPU.

Thus, we will not investigate porting of RAMSES to GPUs any further. Instead, we will focus on CPU optimisations for RAMSES. For GPUs, we will use MINIRAMSES and investigate the use of the SHAMROCK framework to handle the octree structure of the AMR grid.

SHAMROCK (https://shamrock-code.github.io/publications.html) is an in-house framework, under development at CRAL, tailored for numerical astrophysics on exascale architectures. It is based on an accelerated parallel octree and is written in `C++17` with the `SYCL` programming standard to run efficiently on GPUs.

SHAMROCK is currently coupled to an SPH solver (the combination is called SHAMROCK-SPH) and it shows remarkable performance. In Fig. 16, we show a Sedov blast test performed with 65 billion particles. The code has been scaled up to 1024 AMD MI250X GPUs on the Adastra Cluster at CINES in France with a parallel efficiency (weak scaling) of 92% across the entire cluster.

The code is also designed to support AMR grids. We will import the Godunov solver and the PIC module of RAMSES in the octree of SHAMROCK. The first version of SHAMROCK-RAMSES with a 1st-order Godunov solver on a uniform grid already shows promising performance (248,551,348 cell/sec with 256^3 cells per GPU, see Fig. 16). In the next months, we will fully port RAMSES Godunov solver (Kernel 1) to SHAMROCK-RAMSES.

9 Conclusions

This deliverable presents the recent progress in optimising and porting several kernels/modules for improved performance on GPU architectures. The collective efforts across different codes indicate substantial advancements in computational efficiency, scalability, and collaborative development practices. It is worth noting that the implementation of the CI/CD pipeline is a key feature of each of the codes. Any changes committed to the repositories undergo a range of tests to verify compilation, proper executable generation, and running one or more test problems to test correctness and code efficacy have greatly enhanced overall code management. These pipelines ensure that code updates are seamlessly integrated and rigorously tested, promoting a robust and efficient development environment. The development of all of the seven codes within the SPACE CoE can be tracked in the **public**/**open-source** repositories presented in Table 2.

Porting modules to GPU and kernel optimisations have been a key focus, resulting in substantial improvements in computational speed and capability. Compiling and running the codes on different EuroHPC JU clusters such as LUMI, Leonardo, and Karolina is another key achievement. This ensures that the applications can be used in different HPC environments efficiently, making them more accessible and usable for researchers. Recently, most code owners have gotten access to computing time some of the other EuroHPC JU clusters, namely VEGA, MeluXina, MareNostrum, and Discoverer. The applicability of the SPACE codes on these clusters as well as scaling tests will constitute a significant part of the upcoming deliverables.

Code	**Git Repository**	**Public / Open Source**
OpenGADGET	https://code.it4i.cz/space_coe/opengadget3	YES
PLUTO	https://gitlab.com/PLUTO-code/gPLUTO	YES
BHAC	https://code.it4i.cz/space_coe/bhac_mini_app	YES
ChaNGa	https://github.com/N-BodyShop/changa	YES
FIL	https://code.it4i.cz/space_coe/fil	YES
iPic3D	https://code.it4i.cz/space_coe/iPic3D	YES
RAMSES	https://code.it4i.cz/space_coe/ramses	YES

Table 2: List of repository for the codes involved in SPACE CoE.

Overall, the collective efforts in code development highlight the progress of SPACE CoE in facilitating the use of HPC tools for scientific research. The focus on GPU offloading, kernel optimisation, CI/CD implementation, and adaptation to EuroHPC infrastructures has not only improved performance but also fostered a collaborative and innovative environment for ongoing development.

References

[1] 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.

[2] D. Ryu and T. W. Jones, “Numerical Magnetohydrodynamics in Astrophysics: Algorithm and Tests for One-dimensional Flow,” *The Astrophysical Journal*, vol. 442, p. 228, Mar. 1995.

[3] “Repository of the changa code on github,” https://github.com/N-BodyShop/changa, accessed: 28.10.2023.

[4] 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.

[5] 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.

[6] J. G. Stadel, “Cosmological N-body simulations and their analysis,” Ph.D. dissertation, University of Washington, Seattle, Jan. 2001.

[7] S. Markidis, G. Lapenta, and Rizwan-uddin, “Multi-scale simulations of plasma with ipic3d,” *Mathematics and Computers in Simulation*, vol. 80, no. 7, pp. 1509–1519, 2010.

[8] P. J. Deka, L. Einkemmer, and M. Tokman, “LeXInt: Package for exponential integrators employing Leja interpolation,” *SoftwareX*, vol. 21, p. 101302, 2023.

[9] P. J. Deka, A. Moriggl, and L. Einkemmer, “LeXInt: GPU-accelerated Exponential Integrators package,” *arXiv e-prints*, p. arXiv:2310.08344, 2023.

[10] 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.

[11] “Ramses repository on GitLab,” https://bitbucket.org/rteyssie/ramses/src/master/.