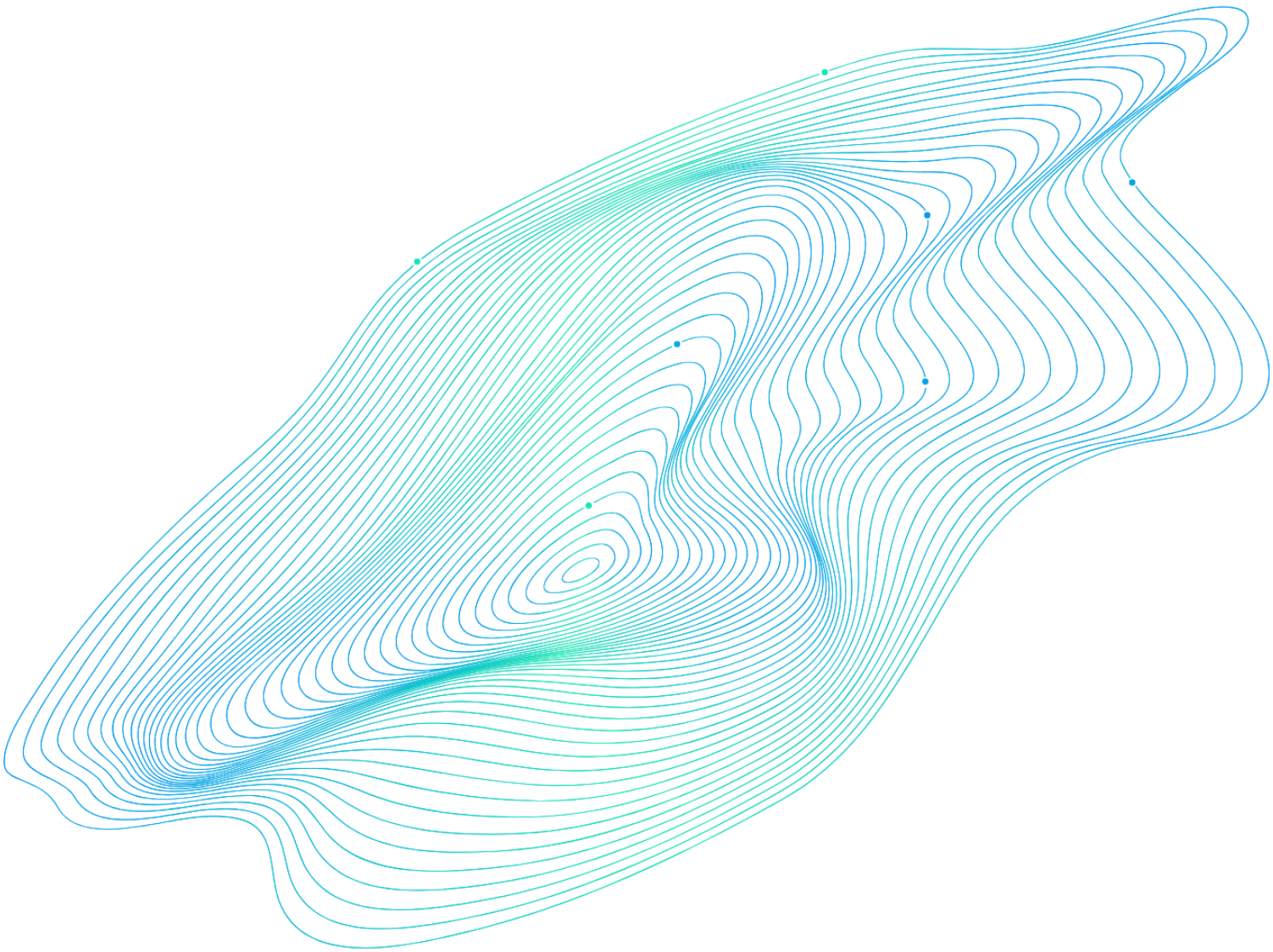
EPICURE

Unlocking European-level HPC Support

HIGH-LEVEL SPECIALISED APPLICATION SUPPORT SERVICE IN HIGH-PERFORMANCE COMPUTING (HPC)



Co-funded by the European Union

EuroHPC Joint Undertaking

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No.101139786. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or EuroHPC Joint Undertaking. Neither the European Union nor the granting authority can be held responsible for them.

EPICURE

DELIVERABLE 2.1

Report on code porting, enabling and scaling for Year 1



Project Title	High-level specialised application support service in High-Performance Computing (HPC)
Project Acronym	EPICURE
Project Number	101139786
Type of Action	DIGITAL JU Simple Grants
Topic	DIGITAL-EUROHPC-JU-2022-APPSUPPORT-01-01
Starting Date of Project	2024-02-01
Ending Date of Project	2028-01-31
Duration of the Project	48 months
Website	epicure-hpc.eu

Work Package	2
Task	2.1, 2.2, 2.3
Lead Authors	David Vicente (BSC-CNS)
Contributors	Ilya Zhukov (JUELICH) Ziyad Albanoby (BSC-CNS) Trevor Hall (BSC-CNS)
Peer Reviewers	Andrew Emerson (CINECA) Joao Barbosa (IT4I) Alberto Lanzanova (CSC)
Version	2.0
Due Date	31/01/2025
Submission Date	29/01/2026 (2nd submission, after review comments)

Dissemination Level

X	PU: Public
	SEN: Sensitive – limited under the conditions of the Grant Agreement
	EU-RES. Classified Information: RESTREINT UE (Commission Decision 2005/444/EC)
	EU-CON. Classified Information: CONFIDENTIEL UE (Commission Decision 2005/444/EC)
	EU-SEC. Classified Information: SECRET UE (Commission Decision 2005/444/EC)

Version History

Revision	Date	Editors	Comments
0.1	10/12/2024	David Vicente (BSC)	Structure of the deliverable
0.2	02/01/2025	David Vicente (BSC)	Full first version completed
0.5	16/01/2025	David Vicente (BSC)	Full version with review comments
1.0	31/01/2025	Eeva Harjula (CSC)	Final quality check performed by the PMO. File ready to be submitted.
1.1	01/11/2025	David Vicente (BSC)	New version including the comments from reviewers based in the M18 review
1.5	30/12/2025	David Vicente (BSC)	Added new project description table, added the applications used in the benchmark suite, references and links to all the document checked.
1.6	09/01/2026	Alberto Lanzanova (CSC) Joao Barbosa (IT4I)	Review of the document content. Correction of few typos, formatting and minor changes to stylistic choices.
1.8	11/01/2026	David Vicente (BSC)	Consolidated version including the comments from the last round of review.
2.0	29/01/2026	Alberto Lanzanova (CSC) Virvatuli Uusimäki (CSC)	Final quality check performed by the PMO. File ready to be submitted.

Glossary of Terms

Item	Description
EuroHPC JU	The EuroHPC Joint Undertaking, a public-private partnership responsible for developing and deploying high-performance computing (HPC) infrastructure across Europe
FOA	Focus of Attention
HPC	High Performance Computing
MN5	MareNostrum 5
MPI	Message Passing interface
OpenMP	OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared-memory multiprocessing programming models
POP	Performance Optimisation and Productivity COE
WP	Work Package

Executive Summary

In the first 12 months, EPICURE has made significant progress in optimizing multiple projects under Work Package 2 (WP2). Efforts have been focused on providing Level 2 support, specifically targeting the optimization of code compilations, workflow efficiency, code analysis, profiling, benchmarking, and job distribution setup.

The work has been structured across three key tasks as outlined in the Description of Actions. At the site level, partners have effectively collaborated, using their collective expertise within WP2 to deliver specialized support to the projects. These efforts have led to measurable improvements in the performance and efficiency of up to 30 projects, enabling better utilization of EuroHPC JU resources.

Table of Contents

1. Introduction 8
2. Task 2.1 - Code Porting and project enabling 9
2.1 Introduction and status 9
2.2 Detailed description per project 12
3. Task 2.2 - Performance improvements, benchmarking and performance analysis 25
3.1 Introduction and status 25
3.2 EPICURE Benchmark Suite 25
3.3 Performance analysis 31
4. T2.3 - Best Practices Guidelines for EuroHPC supercomputers and technical reports 42
4.1 Introduction and status 42
4.2 Best Practices guidelines 42
5. Conclusions 44

List of Tables and Figures

1 – Table with list of projects and their optimization work 11
2 – Figure with the WRF results from the benchmark suite running on MN5 and Leonardo 30
3 – Figure with the CP2K results from the benchmark suite running on MN5 and Leonardo 31
4 – Figure of Nsight Systems trace showing sequential execution of computation and IO 32
5 – Figure of Nsight Systems trace showing successful overlap of computation and IO. 33
6 – Figure of GraphViz visualization profile for project #9 34
7 – Figure of SnakeViz showing execution bottlenecks for project #9 34
8 – Figure Showing GPU utilization metrics 35
9 – Figure Showing GPU source correlation of performance bottlenecks 36
10 – Figure of a summary timeline of the application collected with Score-P on Vega. Executed on 3 nodes (658 MPI ranks) 37
11 – Figure of the number of messages per message size for the total solver loop. 37
12 – Figure of a Zoom-in into five iterations of the solver loop. 38

13 – Figure of the scaling plot of FOA, 10 iterations, 64x64x64 testcase (left), breakdown of execution time (right) 39
14 – Figure of scaling plot of FOA, 10 iterations, 512x512x512 testcase (left), breakdown of execution time (right) 40
15 – Table of POP efficiencies of FOA, 10 iterations, 512x512x512 41
16 – Table of summary of identified inefficient communication patterns of FOA, 10 iterations, 512x512x512 41

1. Introduction

The first months of EPICURE were focused on organizing and structuring the distributed teams in the different EuroHPC JU hosting sites to be ready to provide support for the EuroHPC JU projects. After the first list of projects was received in May 2024, the teams started their work, assigning a lead partner to establish contact with the principal investigators. The first step was to define the expected work and assign the appropriate parties as collaborators from the EPICURE Applications Support Teams (ASTs). The first projects were mainly completed during the May–December period: this document includes the work done in this first set of projects (13 projects initially assigned) and the status of the subsequent batches of projects requiring EPICURE support as provided by the EuroHPC JU. This deliverable will use the WP2 task distribution, as described in the Description of Work, to organize the type of work within the following sections:

- Task 2.1: Code Porting and project enabling
- Task 2.2: Performance improvements, benchmarking and performance analysis
- Task 2.3: Best Practices Guidelines for EuroHPC supercomputers and technical reports

While Task 2.3 is not strictly tied to specific project optimizations, it integrates the findings and outcomes from the previous tasks, ensuring comprehensive reporting and documentation. As such, Task 2.3 has been included in this deliverable, but with a concise scope.

2. Task 2.1 - Code Porting and project enabling

2.1 Introduction and status

The projects enabled during the first 6 months were limited to 13, as they were the ones provided in the first batch from EuroHPC JU. This reduced the usage of resources as was explained in the month six review. The situation has since improved, with more than 30 projects active during the first 12 months, out of a total of 82.

In the Table 1 we have defined each project based on the work required. The different topics are described below the table, with specific descriptions per topic. A final column has been added to indicate projects with WP3 optimizations, consolidating all projects involving optimization work in any of the technical WPs into a single table. However, any detailed information related to WP3 is provided in WP3 deliverables (D3.1).

EPICURE code	EuroHPC CALL	Machine assigned	Title	Enabling	Compilation improvements	Code debug	Benchmarking	Profiling	Performance analysis	Job enabling	Porting	WP3 optimization work
01	Regular Access	Leonardo DCGP	Direct numerical simulation of wall-bounded turbulent flows with adverse and favourable pressure gradients	X	X		X	X			X	
02	Extreme Scale	MareNostrum5 ACC	Design Principles for Non-Aromatic Fluorescence in Biology (DpNAFB)	X		X	X	X	X			X
03	Regular Access	Vega	Anthropogenic Stressors and Future Changes of the North West European Shelf ecosystem	X	X	X	X	X	X			X
04	Extreme Scale	LUMI	Pushing the limits of computational chemistry: the phase diagram of water				X			X	X	
05	Extreme Scale	Leonardo	From methane and iron nanoparticles to "Turquoise" H2 and carbon nanotubes: simulating the whole catalytic process inside a reactor	X	X		X	X	X			X
06	Regular Access	LUMI	Removing cobalt from Hard Metals	X	X		X				X	
07	Regular Access	MareNostrum 5 GPP	RAHMAN		X		X			X		

08	Extreme Scale	LUMI	HiPsum - Translating openQCD-fastsum to AMD	X	X	X		X	X		X	X
09	Regular Access	MareNostrum 5 GPP	AtmoRep	X	X		X	X	X			X
10	Regular Access	Discoverer	Broadband earthquake simulations accounting for source, path and site effects in a 3D velocity model of the Dead-Sea Transform.		X		X					X
11	Regular Access	MeluXina CPU	Large-Eddy Simulations of Supersonic Air Intakes	X	X			X	X	X		X
12	Regular Access	Karolina GPU	Genie 3.0.: an optimized denoising diffusion probabilistic model for protein design.	X			X	X				X
13	Development	Leonardo Booster	Enhancing AI training efficiency and scalability to train EU institutional LLMs for the public sector	X			X			X	X	X
16	Regular Access	MareNostrum5 ACC	DEAREL: Large-scale Deep rEinforcement leARning for activE fLow control in wings				X	X	X			X
24	AI	MareNostrum5 ACC	Developing Video-Based Language Models for Time Study and Value Analysis in Manufacturing	X						X	X	
25	AI	Leonardo Booster	Learning Object-Centric Representations for Self-Driving		X					X		
26	AI	LUMI-G	Ai4Agriculture				X	X				
29	AI	MareNostrum5 ACC	Investigation of novel epigenetic signals through combinatorial crosstalk between histone isoforms	X			X			X	X	
32	AI	Leonardo Booster	CAPHARD: Codon-Assisted Phage-Host Automatic inteRaction Discovery	X			X			X		
38	Development	MareNostrum5 ACC	Learning visual foundation models from videos	X			X			X		
48	Benchmark	MeluXina GPU, Karolina GPU, LUMI-G, MareNostrum5 ACC	Cross-Facility Federated Learning	X	X		X					
50	Development	LuxProvide	Machine Learning of Exchange Energy Density and Integration with Python-based Electronic Structure Codes	X			X	X				X

EPICURE code	EuroHPC CALL	Machine assigned	Title	Enabling	Compilation improvements	Code debug	Benchmarking	Profiling	Perf analysis	Job enabling	Port ing	WP3 optimization work
51	Development	LUMI-G, Leonardo Booster	Enhanced scalability in EPSILOD: A framework for generic iterative loop stencil applications	X	X		X		X			
52	Extreme Scale	LUMI	High-throughput discovery of magnetic molecules for quantum technologies	X	X		X				X	
53	Extreme Scale	MareNostrum5 ACC	Advancing RNA Therapeutics: Molecular Dynamics Insights into Lipid Nanoparticle Design and Optimization for Enhanced Delivery	X			X			X		
55	Extreme Scale	Leonardo Booster	Curriculum Learning for large language models in low-resource languages	X			X			X	X	
57	Extreme Scale	Leonardo Booster	Quantum Monte Carlo generation of machine learning potentials for hydrogen-based high-temperature superconductors	X	X	X	X					
60	Extreme Scale	Leonardo Booster	2D materials modelling via QMC-quality Machine Learning Potentials	X		X	X				X	X
66	Benchmark	MareNostrum5 ACC	On Device AI Assistant: Revolutionizing Real-time API Interactions with Privacy Preserving Intelligence	X			X			X	X	X
67	Development	Discover, Leonardo, Vega	HIDALGO 2 Wildfires deployment 2024	X	X	X					X	

1 – Table with list of projects and their optimization work

Here are the descriptions for each type of optimization listed:

Enabling: This involves preparing the software or tools required for the project to work in the selected machine/architecture. It also includes tasks like verifying software dependencies, ensuring compatibility with the hardware, and addressing system-specific requirements to make the application operational.

Compilation Improvements: This focuses on enhancing the build process of the software required by the project, in order to optimize the performance on the target system. Tasks include using advanced compiler flags, selecting the best compilers for the architecture, the best mathematical libraries, improvements in code vectorization at the compilation level, or adding specific flags to produce highly efficient binaries.

Code Debugging: Debugging is an important aspect of moving a project to production in a new system and implies identifying and proposing solutions for errors or bugs in the code. This optimization ensures that the software runs correctly and efficiently.

Benchmarking: Benchmarking measures the performance of the software used by the projects in the system or architecture selected. It is an important because it ensures that an optimization activity has had a positive impact on the performance.

Profiling: Profiling involves analysing the behaviour of an application to identify performance-critical sections.

Performance Analysis: The Performance analysis is an extension of the profile to generate a deep analysis of the full application, including MPI communications patterns, Input/Output, hardware counters to understand better each fraction of the code and the corresponding analysis of the code traces to provide this final information to the developers with a series of recommendations to improve the code.

Job Enabling: This part describes the work in optimizing job scripts and configurations to maximize resource usage and minimize execution time. It includes tasks like fine-tuning job scheduling parameters, setting memory and CPU affinity, and optimizing NUMA domain usage.

Porting: Porting involves adapting software to run on a different architecture, operating system, or hardware platform. This includes addressing differences in compilers, libraries, or system calls and modifying the codebase to leverage the unique features of the new environment.

2.2 Detailed description per project

This section provides a detailed account of the activities and outcomes of each project. For each project, the summaries below include a brief description highlighting their focus, key activities undertaken, the resulting outcomes and a conclusion.

Project 1	**EHPC-REG-2023R02-115**	Lead Partner: CINECA
Machines: Leonardo		Collaborators: --
Project Goals	The goal was to port, compile, and optimize the TBLDNS code for the Leonardo DGGP partition, ensuring it performs efficiently on Intel Sapphire Rapids CPUs and integrates properly with Intel oneAPI libraries.	
Work done	The team compiled the code using Intel oneAPI MPI and MKL instead of OpenMPI and FFTW. They tested two configurations (a large case on 40 nodes and a smaller single-node case). Custom optimization flags were applied: -O2 -qopenmp -align array64byte -heap-arrays 4096 -xCORE-AVX512 -qopt-zmm-usage=high -fp-model fast=2 -complex-limited-range.	

	Performance measurements were carried out using TBLDNS profiling tools and Intel Application Performance Snapshot.
Improvements provided	• Code performance improved from 50.44s to 47.73s per time step. • Double precision performance increased from 113.88 to 120.03 GFLOPS. • Identified that the application is memory-bound, confirming optimizations were effective.
Conclusions	The project successfully optimized TBLDNS for Intel architectures on Leonardo, achieving measurable performance gains and providing a clear set of compiler flags beneficial for other HPC applications on similar platforms.

Project 2	**EHPC-EXT-2023E01-029**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Optimize and analyse the performance of the wbse_init and wbse.x programs in the DpNAFB workflow, with a focus on GPU performance, reduction operations, and asynchronous I/O handling.	
Work done	Used Nvidia Nsight Systems to profile GPU kernels and identify bottlenecks in reduction operations, accounting for up to 78% of runtime. Created C bindings to pthreads to bypass nvfortran segmentation faults and developed a thread pool for async I/O. Loop unrolling and reordering were applied in OpenACC, later mapped to Thrust C kernels through a Python-C interface.	
Improvements provided	• Detailed profiling enabled isolation of heavy GPU kernels. • Threaded async I/O partially improved data throughput. • OpenACC kernel optimization yielded measurable performance benefits.	
Conclusions	Profiling and kernel analysis provided actionable insights that were used in the WP3 work and provided performance gains. While certain Fortran async I/O approaches failed, hybrid C and Python integration proved effective for enhancing GPU utilization.	

Project 3	**EHPC-REG-2023R02-047**	Lead Partner: IZUM
Machines: Vega		Collaborators: FZJ
Project Goals	Improve performance and scalability of the ecosystem simulation code for the Northwest European Shelf under anthropogenic stressors and climate change.	
Work done	Performance optimization was carried out using SLURM CPU pinning flags to efficiently utilize CPU cores. Multiple benchmark runs were conducted to evaluate and visualize the resulting speedups. Additionally, a detailed performance analysis was performed by FZJ to identify potential bottlenecks and areas for improvement in the code, using Score-P as the profiling tool.	
Improvements provided	• Significant runtime improvements (Up to 7% speedup of running real-life multi-node example) achieved through better CPU pinning strategies. • Demonstrated strong scaling behaviour on Vega nodes.	

Conclusions	Targeted low-level configuration improvements (SLURM pinning) yielded meaningful speedups, demonstrating that simple scheduler-level tuning can significantly enhance performance for large-scale environmental simulations.

Project 4	**EHPC-EXT-2023E02-065**	Lead Partner: UAntwerpen
Machines: LUMI		Collaborators: --
Project Goals	Develop multi-node capability for large-scale phase diagram simulations of water, focusing on efficient task throughput and inter-node communication using i-PI and Parsl.	
Work done	Planned to extend i-PI client communication from UNIX to INET sockets to support multi-node computations. Initial discussions and feasibility analysis on scaling up from single-node to multi-node setups using CP2K, ORCA, PLUMED, PyTorch, and Parsl. Project paused due to PI leaving the institution.	
Improvements provided	Preliminary design for multi-node communication extension in i-PI and strategies for optimizing filesystem usage and task throughput were prepared.	
Conclusions	Project setup and technical planning were completed, but implementation was halted. The groundwork remains available for continuation by a future lead investigator.	

Project 5	**EHPC-EXT-2023E01-036**	Lead Partner: CINECA
Machines: Leonardo		Collaborators: --
Project Goals	Modernize Python code and perform profiling to detect performance bottlenecks and optimization opportunities.	
Work done	• Updated Python environment from 3.8.8 to 3.12 and major libraries (NumPy, SciPy, Pymatgen, Atomate, Fireworks). • Profiled RBF interpolation unit tests and implemented extended unit tests to reproduce bottlenecks. • Used Scalene and cProfile for thread-level performance analysis, including those spawned by NumPy/SciPy.	
Improvements provided	• Environment modernization enabling compatibility with new Python and libraries. • Identification of threading and interpolation bottlenecks. • Better monitoring of Python thread usage.	
Conclusions	Despite profiling tool limitations on Python 3.12, the project achieved full modernization of its computational stack and improved performance visibility, setting a foundation for more efficient catalytic process simulations.	

Project 6	**EHPC-REG-2023R03-183**	Lead Partner: KTH
Machines: LUMI		Collaborators: INESC TEC
Project Goals	Port and optimize simulation codes (VASP, LAMMPS, and PACEmaker) to LUMI's CPU (LUMI-C) and GPU (LUMI-G) partitions, addressing I/O bottlenecks and ensuring efficient scaling for hard metal material simulations.	
Work done	• Installed and customized VASP 6.3.2 and 5.4.4, patched to reduce I/O load.	

	• Built LAMMPS with Kokkos and GPU support; benchmarked on LUMI-C and LUMI-G. • Created a containerized Python software stack to overcome file system issues. • Worked with LUMI Support Team to adjust disk policies and enable hybrid job submission. • Performed benchmarking and hyperparameter tuning (especially KPAR optimization).
Improvements provided	• Achieved 3–4× speed-up with GPU-accelerated LAMMPS vs CPU LAMMPS (LUMI-G vs LUMI-C). • Identified optimal KPAR values for metallic systems (e.g., KPAR=13 for 455 k-points). • Demonstrated that not all cores need to be used for maximum VASP speed (in this case, maximum speed was reached with 91 of 128 cores used due to a better fit with k-point parallelization.)
Conclusions	Efficient porting and performance optimization were achieved. GPU-accelerated LAMMPS and improved k-point parallelization in VASP provided cost-effective speedups for material science applications on LUMI.

Project 7	**EHPC-REG-2023R02-125**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Optimize GROMACS molecular dynamics simulations for the RAHMAN Project on MN5 infrastructure, improving efficiency and scalability for large biological systems used in rare disease research.	
Work done	• Profiled and optimized PP/PME rank distribution and Dynamic Load Balancing (DLB). • Evaluated hyperthreading, thread affinity, and compilation options (AVX512 performed best). • Conducted scaling and benchmarking tests across up to 10 nodes.	
Improvements provided	• Reduced PP/PME imbalance from 6.9% to 0.1% and load imbalance from 3.5% to 1.7%. • Achieved up to 5% speedup with hyperthreading in single node jobs. • Improved multi-node scalability while maintaining stable performance.	
Conclusions	Single-node runs were the most efficient; scaling to 10 nodes improved runtime but reduced resource efficiency. Hyperthreading benefited small runs but not multi-node setups. Overall, GROMACS performance on MN5 was significantly optimized, enhancing simulation throughput for the RAHMAN project.	

Project 8	**EHPC-EXT-2023E01-010**	Lead Partner: CSC
Machines: LUMI		Collaborators: --
Project Goals	Translate openQCD-fastsum from CUDA to AMD HIP for LUMI, improving portability and maintainability.	

Work done	• Migrated build system from manual Makefiles to CMake for easier deployment. • Used hipify scripts to convert CUDA code to HIP, with minor manual fixes.
Improvements provided	• Successful partial translation and modernization of the codebase. • Simplified build process via CMake, improving future portability.
Conclusions	Despite limited engagement from the PI and missing test cases, core GPU porting was completed. The project highlighted the importance of clear communication and validated the robustness of HIP conversion tools.

Project 9	**EHPC-REG-2023R03-078**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Profile and optimize the AtmoRep model for atmospheric representation learning, focusing on I/O and data loading performance in PyTorch.	
Work done	• Installed and ran AtmoRep using 2010–2020 data for training, 2021 for validation. • Profiled with cProfile, SnakeViz, and GraphViz; debugged with DebugPy. • Identified MultifieldDataSampler.iter() and NumPy's take() as key bottlenecks.	
Improvements provided	• Isolated high-cost routines for refactoring. • Gained insights into data-loading inefficiencies. • Collected performance profiles for optimization guidance.	
Conclusions	Extensive profiling pinpointed computational hot spots, laying groundwork for WP3 code restructuring. Some debugging tools failed with multiprocessing, revealing the need for framework-level data loader redesign.	

Project 10	**EHPC-REG-2023R2-138**	Lead Partner: SofiaTech
Machines: Discoverer		Collaborators: --
Project Goals	Enable CUDA-aware RAJA integration into the seismic simulation code SW4 for broadband earthquake modelling in 3D.	
Work done	• Configured environment on Discoverer (GCC 12.3.0, CUDA 12.3). • Integrated external libraries (HDF5, LAPACK, FFTW, UCX). • Compiled SW4 (raja branch) using patched Makefile.nvcc due to missing CMake support.	
Improvements provided	• Established reproducible build recipes and improved compatibility tracking. • Fixed build issues and enabled GPU support with patched build process.	
Conclusions	The project successfully built a working CUDA-aware SW4 version with RAJA support but identified outdated documentation and poor integration practices. Future work should standardize the build process with CMake.	

Project 11	**EHPC-REG-2023R03-113**	Lead Partner: LUXPROVIDE SA
Machines: Meluxina		Collaborators: FZJ
Project Goals	Optimize the IC3 solver for large-eddy simulations of supersonic air intakes by improving vectorization, cache efficiency, and communication patterns.	

Work done	The IC3 solver was analysed and optimized for performance. Compilation flags were tested to identify optimal CPU settings, with improvements of 3–4% over the base compilation scenario. Profiling was performed using Linaro-map, Extrae, Score-P, and Maqao, revealing that indirect memory accesses limited vectorization to 47%, with numerous cache misses caused by cell and face traversal. MPI profiling was conducted by Jülich Supercomputing Centre (JSC) to ensure no communication or synchronization issues, using Score-P for measurements and Vampir for trace visualization. Multi-node scalability of the IC3 solver was confirmed, though a Late Sender Wrong Order communication pattern was observed. It was recommended to replace MPI_Irecv source parameters with MPI_ANY_SOURCE. Process affinity was explored to reduce cache misses, using SLURM and numactl for binding tasks to threads, cores, and sockets. Binding tasks to the two logical threads of CPU cores provided slightly better performance than other binding options.
Improvements provided	• Compiler flags optimization: Small improvements of 3–4% over base compilation. • Process affinity and binding strategies: Enhanced cache utilization and memory locality. • MPI analysis and recommendations: Identified and mitigated communication issues to improve code stability and multi-node behaviour.
Conclusions	• Vectorization remains limited to 47% due to indirect memory access. • Cache misses during cells and faces traversal remain a primary performance bottleneck. • Process affinity and careful task-to-core mapping provide performance benefits. • Multi-node scalability is confirmed, with no major MPI synchronization issues after applying recommended fixes.

Project 12	**EHPC-REG-2023R03-016**	Lead Partner: IT4I@VSB
Machines: Karolina		Collaborators: --
Project Goals	Deploy and scale the Genie 3.0 diffusion model for protein design on EuroHPC JU infrastructure, ensuring environment reproducibility and scalability.	
Work done	• Deployed Genie using PyTorch Lightning on Karolina GPUs. • Created isolated environments, integrated with Slurm, and optimized memory. • Addressed dependency issues (CUDA, wandb, Horovod) and lack of containerization.	
Improvements provided	• Achieved linear scalability up to 128 GPUs. • Optimized memory and runtime via torch.compile() and padding removal (Torch compilation reduced runtime by approximately 33%.) • Demonstrated batch parallelism as a viable alternative to Horovod which gave problems due to some incompatibilities with Karolina software stack.	

Conclusions	Genie was successfully deployed and scaled. The project demonstrated effective scaling and provided significant improvements using torch.compile.

Project 13	**EHPC-DEV-2024D05-041**	Lead Partner: CINECA
Machines: Karolina		Collaborators: --
Project Goals	Enable multi-node, multi-GPU distributed training using PyTorch Distributed in Singularity containers on Leonardo, integrate with JUBE for benchmarking and profiling, and assess performance of large models up to 70B parameters.	
Work done	• Modified job scripts to support multi-GPU, multi-node runs via torchrun and SLURM integration. • Integrated the workflow with JUBE, profiling with nsys and DeepSpeed. • Benchmarked models (13B–70B) using LlamaForCausalLM + DDP + DeepSpeed, confirming NCCL as communication backend. • Adopted Mixtral with ColossalAI plugins, enabling CPU offload, pipeline parallelism, and reduced sequence length to handle high memory use. • Investigated NVMe offload issues linked to a known DeepSpeed bug.	
Improvements provided	• Achieved scalable multi-node training with efficient GPU utilization. • Reduced GPU memory footprint through CPU offload and pipeline tuning. • Ensured reliable profiling and benchmarking across large model configurations.	
Conclusions	The setup deployed in the project successfully enabled large-scale distributed training on Leonardo using singularity containers and slurm as base. Mixtral provided similar performance to larger models with lower cost, while CPU offload and pipeline tuning were crucial to manage memory constraints.	

Project 16	**EHPC-REG-2024R01-038**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Profile and optimize deep reinforcement learning workloads for active flow control in aerodynamics (DEAREL project).	
Work done	• Used NVIDIA Nsight Systems to trace 30 time steps of the simulation. • Identified synchronization and redundancy issues in GPU kernel calls (30 vs 600 function calls). • Attempted to replace global atomics with sort + reduce_by_key, reverted due to numerical constraints.	
Improvements provided	• Identified excessive kernel launches per time step. • Collected detailed timing data enabling optimization planning.	
Conclusions	Comprehensive profiling revealed fine-grained performance issues in iterative GPU kernels. Although algorithmic constraints limited atomic optimizations, the profiling data guided kernel restructuring and optimizations done in the WP3.	

Project 24	**EHPC-AI-2024A01-034**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Enable the use of advanced video-language AI models on MareNostrum5 (MN5), including setup, dependency installation, and initial testing.	

Work done	• Ported the user's Python environment to MN5. • Installed and configured several video-language models: CogVLM2, MovieChat, Qwen2VL, VideoLLaMA, and VideoLLaVA. • Assisted the user with initial runs and explained MN5-ACC partition setup for AI workloads.
Improvements provided	• Enabled rapid deployment of multiple AI video-language models. • Reduced setup time for users by providing working configurations and example runs.
Conclusions	Initial setup completed successfully; users could run models independently afterward. Limited ongoing support was required due to the straightforward deployment goals.

Project 25	**EHPC-AI-2024A01-060**	Lead Partner: CINECA
Machines: --		Collaborators: --
Project Goals	Develop object-centric representations for self-driving applications using AI on EuroHPC JU systems.	
Work done	No technical work performed, project was archived at PI request before implementation began. The project was initially included in the M12 report as ongoing but was later archived following the PI request. No efforts were expended or reported.	
Improvements provided	None (archived).	
Conclusions	Project closed by PI before start of technical activities.	

Project 26	**EHPC-AI-2024A01-080**	Lead Partner: KTH
Machines: --		Collaborators: --
Project Goals	Leverage AI methods for agriculture-related analysis and prediction.	
Work done	Basic analysis of the project completed. No technical work performed, project was archived at PI request before implementation began. The project was initially included in the M12 report as ongoing but was later archived following the PI request. Minimum efforts were expended or reported, just for the basic analysis of the project.	
Improvements provided	None (archived).	
Conclusions	Project closed by PI before start of technical activities.	

Project 29	**EHPC-AI-2024A01-088**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Accelerate GROMACS simulations studying epigenetic signals by enabling GPU usage and multi-node performance scaling.	
Work done	• Adapted CPU-only GROMACS scripts for GPU acceleration. • Compiled GROMACS 2024.2 with CUDA, MPI, and OpenMP support.	

	• Manually mapped MPI ranks to CUDA devices using NUMA binding. • Enabled NVIDIA MPS to improve GPU concurrency. • Tested strong-scaling limits on 1–5 MN5 ACC nodes (up to 20 GPUs)
Improvements provided	• A speedup of about 1.6 was achieved after the tuning work done by the EPICURE team, when scaling from 1 to 2 nodes (4–8 GPUs), which is acceptable for production runs. • GPU acceleration provided major gains compared to CPU-only runs. • Manual MPI–GPU mapping proved critical for scaling.
Conclusions	Successful GPU acceleration performance achieved; however, scaling beyond two nodes was very limited using the case provided by the user. NVIDIA MPS offered minor benefit for multi-node runs.

Project 38	**EHPC-DEV-2024D08-011**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Deploy environment and training setup for video-based foundation models on MN5-ACC.	
Work done	• Configured training environment for visual models. • Provided basic training instructions and environment validation. • User trained initial image model successfully.	
Improvements provided	• Completed user environment setup and basic training enablement.	
Conclusions	Minimal intervention, project focused on initial setup and user onboarding. No further optimization requested.	

Project 48	**EHPC-DEV-2024D08-011**	Lead Partner: LUXPROVIDE SA
Machines: Meluxina		Collaborators: --
Project Goals	Scale and optimize federated learning workloads across multiple facilities.	
Work done	• Waiting for PI's updated code before profiling. • Conducting scaling experiments on Meluxina using models similar to PI's architecture.	
Improvements provided	• Early scaling insights for cross-facility training.	
Conclusions	Support ongoing; awaiting updated PI code for full optimization phase. The project is scheduled for development over the next 12 months, with any progress to be included in the upcoming deliverable planned for M24.	

Project 50	**EHPC-DEV-2024D07-076**	Lead Partner: LUXPROVIDE SA
Machines: Meluxina		Collaborators: UAntwerpen
Project Goals	Improve scalability and memory efficiency in Python code for exchange energy density calculations.	
Work done	• Profiled code with Linaro Forge Map to identify tensor contraction bottlenecks.	

	• Explored multi-processing approaches and reduced problem sizes for testing.
Improvements provided	The profile generated provide enough information to focus efforts in WP3 developments, around dask parallelization for the code and memory reductions on the tensor contraction part.
Conclusions	Memory and parallel performance significantly improved. Code now distributed and scalable, pending final verification from the PI.

Project 51	**EHPC-DEV-2024D07-079**	Lead Partner: UAntwerpen
Machines: LUMI		Collaborators: --
Project Goals	Enhance scalability and portability of the EPSILOD framework for iterative stencil computations.	
Work done	• Ported code to LUMI and fixed multiple CMake and compiler issues. • Installed AdaptiveCpp and Kokkos with HIP support. • Created EasyBuild recipes for reproducibility. • Profiled code using Rocprof + Perfetto. • Investigated CPU-GPU binding and transfer bottlenecks.	
Improvements provided	• Fixed memory placement performance issues. • Demonstrated hipcc compiler produced faster code than amdclang. • Provided EasyBuild configs for reproducible setup.	
Conclusions	Code successfully ported and optimized for LUMI; profiling revealed data-transfer inefficiencies. Project now reproducible and maintainable via EasyBuild.	

Project 52	**EHPC-EXT-2024E01-062**	Lead Partner: DeiC (DTU)
Machines: LUMI		Collaborators: --
Project Goals	high-throughput discovery of magnetic molecules for quantum technologies by optimizing job management and MPI usage.	
Work done	• Evaluated OpenMPI behaviour on LUMI. • Recommended use of Hyperqueue for multi-job scheduling.	
Improvements provided	• Introduced scalable job distribution strategy for variable-length tasks.	
Conclusions	Identified viable concurrent job management strategy for heterogeneous workloads using Hyperqueue.	

Project 53	**EHPC-EXT-2024E01-045**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Optimize molecular dynamics simulations for RNA therapeutics and provide guidance for Alphafold2 deployment.	
Work done	• Tuned MPI/OpenMP balance for GROMACS runs. • Enabled NVIDIA MPS and manual MPI rank–GPU mapping. • Developed wrapper scripts for NUMA binding. • Provided baseline setup for Alphafold2 Singularity container.	
Improvements provided	• Efficient GPU utilization through manual mapping. • PME components successfully offloaded to GPUs.	

	• Validated Alphafold2 container deployment.
Conclusions	GPU performance improved through manual PP/PME distribution and MPS. MPI auto-balancing less effective; manual tuning essential for optimal runs.

Project 55	**HPC-EXT-2024E01-042**	Lead Partner: CINECA
Machines: Leonardo, Vega		Collaborators: IZUM, JSI
Project Goals	Train large language models (LLMs) for low-resource languages using NVIDIA NeMo framework.	
Work done	• Supported multi GPU training on Vega and Leonardo. • Debugged container CUDA binding issues (libcuda.so detection). • Resolved environment path and binding variables for the container image for NeMo and Triton. • Enabled successful training after some container fixes.	
Improvements provided	• Fixed CUDA binding issue preventing containerized training on Leonardo. • Provided required environment variable setup for NeMo and Triton containers.	
Conclusions	Successfully completed training after container corrections. Demonstrated importance of proper GPU driver bindings for prebuilt frameworks across different clusters.	

Project 57	**EHPC-EXT-2024E01-064**	Lead Partner: CINECA
Machines: Leonardo		Collaborators:
Project Goals	Support TurboRVB for Quantum Monte Carlo simulations on GPUs, identifying and resolving CUDA incompatibilities.	
Work done	• Reproduced and fixed CUDA header incompatibilities (cublas.h vs cusolverDn.h). • Compiled TurboRVB on Leonardo Booster (CUDA 11.8). • Profiled with Nsight Systems; identified small kernel overheads and CPU inefficiencies. • Suggested multi-process-per-GPU execution via NVIDIA MPS; provided Slurm templates.	
Improvements provided	• Demonstrated MPS achieves similar performance with ¼ the GPU resources. • Reported optimization opportunities to TurboRVB developers.	
Conclusions	GPU-enabled TurboRVB successfully ported. MPS proved highly effective for embarrassingly parallel workloads; OpenMP offload in Fortran noted as unreliable as it causes wrong behaviour of the fortran intrinsic “MAXVAL” (reproducer sent to Nvidia for its fix).	

Project 60	**EHPC-EXT-2024E01-076**	Lead Partner: CINECA
Machines: Leonardo		Collaborators: IT4I@VSB, DeiC (DTU)

Project Goals	Develop machine-learning potentials for 2D materials with QMC-level accuracy.
Work done	• First analysis done about the possible project optimizations. • Awaiting benchmarking and performance testing phases from the PI.
Improvements provided	Support planned but not yet active, awaiting project feedback after access approval.
Conclusions	First analysis done, but technical support not yet started. Awaiting feedback from the PI.

Project 66	**EHPC-BEN-2024B10-067**	Lead Partner: BSC
Machines: MareNostrum 5		Collaborators: --
Project Goals	Deploy privacy-preserving on-device AI assistants using RAY for distributed inference with large language models (VLLM).	
Work done	• Implemented RAY cluster for manual resource control to synchronize with SLURM multi-node jobs. • Converted VLLM Docker environment to Singularity for MN5 compatibility. • Tested simultaneous parallel inference calls to evaluate production throughput (tokens/s)	
Improvements provided	• Script detection system based on resource specification by slurm for multimode RAY cluster distribution. • Using aggregate tensor-parallel configuration for improved scalability. • Running VLLM in Python environment (non-containerized) caused instability in large runs.	
Conclusions	Successfully scaled VLLM via RAY on MN5 using Singularity containers. Demonstrated strong scaling and throughput under parallel generation, enabling larger on-device AI model deployments.	

Project 67	**EHPC-DEV-2024D09-058**	Lead Partner: SofiaTech
Machines: Discoverer, Karolina, LUMI, MareNostrum 5, MeluXina, Vega		Collaborators: UAntwerpen, LUXPROVIDE SA, JSI, IZUM, BSC, SofiaTech, IT4I@VSB
Project Goals	The main goal of this project is to deploy and optimize the HIDALGO 2 Wildfire Simulation Framework across multiple EuroHPC JU systems. The focus is on establishing a unified, high-performance computational environment that can run large-scale wildfire simulations efficiently using modern Fortran compilers and optimized CPU architectures.	
Work done	• System Environment Setup: • Baseline environments were prepared using GCC (8.5.0 and 12.3.0) and Flang (flang-new 19.1.5, AOCC 5) compilers. • Compiler Optimization: • Preliminary investigation into LTO (Link-Time Optimization) support and SIMD instruction tuning (AVX2, AVX512) for computational acceleration. • LLVM/MLIR Integration:	

	• Set up for advanced optimization passes through LLVM Bold/MLIR to enhance compiled Fortran code performance. • External Software Components: • Integration and build of multiple dependencies, including: o Zlib, XZ, LLVM, Zstandard, UCX, OpenMPI, FFTW, HDF5, LAPACK/BLAS o Evaluation of BLIS as an alternative BLAS provider. • Version Control and Build Reproducibility: • Each component is built using dedicated build recipes to ensure reproducibility across EuroHPC JU systems.
Improvements provided	• Standardized multi-site deployment environment for wildfire simulation software. • Enhanced compiler performance insights across GCC, Flang, and LLVM toolchains. • Improved code reproducibility and cross-site portability using centralized build recipes.
Conclusions	The project is in progress, with substantial groundwork completed in compiler setup, environment configuration, and dependency management. Next steps include detailed performance benchmarking, cross-site validation, and finalizing the simulation framework deployment on all listed EuroHPC JU systems.

3. Task 2.2 - Performance improvements, benchmarking and performance analysis

3.1 Introduction and status

This task was divided in two parts reflecting different aspects of the work:

1. the efforts related to the EPICURE benchmark suite, which is not directly tied to any specific EPICURE project but is highly relevant as it provides valuable insights into application performance, scalability, and the status of EuroHPC JU systems (see section 3.2);
2. the performance analysis and profiling work directly linked to EPICURE projects (see section 3.3).

The first part focuses on the EPICURE benchmark suite, while the second part highlights examples of performance analysis and profiling conducted on EuroHPC JU projects with EPICURE support.

The benchmark suite has completed its design and initial implementation phases and is currently undergoing testing at two EPICURE sites: BSC and CINECA.

For performance analysis, several projects have required this type of support. The current status includes examples such as Python debugging, CUDA debugging at two levels (kernel and application), and comprehensive application analysis for MPI/OpenMP codes.

3.2 EPICURE Benchmark Suite

3.2.1 Introduction

The EPICURE Benchmark Suite is a tool designed to automate the installation, benchmarking, and performance analysis of scientific software on high-performance computing (HPC) systems. It aims to streamline the process of assessing software scalability, optimizing resource usage, and identifying performance bottlenecks. The suite integrates multiple tools and frameworks, including EasyBuild[1], JUBE[2], Extrae[3], Paraver[4], and TALP[5], to create a unified system that can be easily deployed across different EuroHPC JU environments.

[1] EasyBuild automatic installation tool, https://easybuild.io/
[2] JUBE Benchmarking tool, https://github.com/FZJ-JSC/JUBE
[3] Extrae, tracer from BSC performance tools, https://tools.bsc.es/extrae
[4] Parver, visualizer from BSC performance tools, https://tools.bsc.es/paraver
[5] TALP, module used for monitor efficiencies, https://pm.bsc.es/ftp/dlb/doc/user-guide/how_to_run_talp.html

3.2.2 Design

The EPICURE Benchmark Suite is designed to work in a highly modular and flexible manner. It combines the following components:

- EasyBuild: Automates the installation and configuration of scientific software, ensuring that dependencies are correctly managed and that reproducible environments are created.
- JUBE: Executes benchmarking applications and collects performance metrics, automating the entire benchmarking workflow.
- Extrae and Paraver: Used to trace and visualize the performance of applications, identifying bottlenecks and inefficiencies in both single-node and multi-node configurations.
- TALP: Gathers real-time performance data for MPI applications, helping analyse parallelism and communication patterns.

The system is designed to be adaptable to a variety of HPC systems, including both CPU and GPU partitions, with a focus on scalability and user accessibility. The automation of tasks like installation and benchmarking ensures that even researchers with limited HPC experience can effectively use the suite.

3.2.3 Implementation

The EPICURE Benchmark Suite is implemented as a Python-based framework that integrates the various tools and provides an easy-to-use interface for researchers. Key features of the implementation include:

- Automation of Software Installation: EasyBuild is configured to handle the installation of scientific applications and their dependencies across different architectures, reducing manual configuration efforts.
- Benchmarking Automation: JUBE scripts manage the execution of benchmark tests, allowing the system to automatically collect performance data and generate results without requiring manual intervention.
- Performance Tracing: Extrae and Paraver are integrated into the suite to capture detailed performance data during benchmarking, enabling the visualization of application behaviour and identification of performance bottlenecks.
- Real-Time Data Collection: TALP is used to gather live performance metrics, offering insights into parallel performance and system utilization.

The implementation is designed to be easily extendable, allowing the EPICURE partners to add new applications and benchmarks as needed.

The current version of the benchmark suite is at BSC gitlab but will be moved to EPICURE gitlab as soon as all the results from MN5 and Leonardo are completed. The current git link is the following: https://gitlab.bsc.es/support/epicure-benchmarks/

3.2.4 Applications

For the EPICURE benchmark applications, we have selected a diverse set of codes from multiple sources. These include applications widely used in high-performance computing (HPC), European codes featured in several Centres of Excellence (CoEs), codes previously included in benchmark suites such as PRACE UEABS, and those used in past HPC procurement processes, including the MareNostrum5 evaluation.

The initial set of applications was defined at an early stage to demonstrate the benchmarking process, while remaining open to incorporating new codes and inputs. In this section, we present the current list of benchmarks included in EPICURE. As part of our future work, we aim to expand this set by adding applications and input cases that can scale to pre-exascale and exascale systems. This will highlight the value of the EPICURE benchmarks not only for current petascale systems but also for next-generation, highly scalable architectures.

3.2.4.1 ALYA

ALYA is multi-physics simulation code developed at BSC. ALYA was designed from scratch as a multi-physics parallel code which solves discretized partial differential equations (PDEs). Here parallelization is based on mesh partitioning (using METIS and MPI tasks). It is written in FORTRAN 90.

Due to limited access to the ALYA source code, this benchmark will only be run on certain systems. We are collaborating with the ALYA developers to resolve this issue, and if a solution cannot be found, the benchmark will be removed from our list.

Test Dataset used

sphere-16M: This input file is extracted from the gitlab repo provided by the developers of ALYA. It is a medium size input that can scales to thousands of cores. It is a good example of a multi-physics simulation to scale in a hybrid mode including OpenMP and MPI parallelization.

Dataset characteristics:

- Mesh: 16.7M elements, 2.9M nodes
- Module used: Nastin
- low over a sphere
- Conjugate Gradient algebraic solver method used

3.2.4.2 CP2K

CP2K is a quantum chemistry and solid-state physics software package that can perform atomistic simulations of solid state, liquid, molecular, periodic, material, crystal, and biological systems.

Test Dataset used

H2O-DFT-LS.inp: This input file is extracted from the [Unified European Applications Benchmark Suite (UEABS)] CP2K dataset. It was chosen because it is a standard benchmark case commonly used to evaluate performance and scalability.

Dataset characteristics:

- Performs a single-point energy calculation of 2048 water molecules
- Uses linear-scaling Density Functional Theory (DFT).
- Contains a total of 6144 atoms.

3.2.4.3 GROMACS

GROMACS is a versatile package to perform molecular dynamics, i.e. simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for simulating proteins, lipids, and nucleic acids.

Test Dataset used

lignocellulose-rf.tpr: This input file is taken from the [Unified European Applications Benchmark Suite (UEABS)] GROMACS dataset. It was selected because it models a heterogeneous lignocellulose system (spatially inhomogeneous) using Reaction-field electrostatics instead of PME, which is less communication-intensive and therefore shows better scalability for large number of nodes. For instance, according to PRACE-2IP data, this benchmark demonstrated good scaling up to 10,000 cores.

Dataset characteristics:

- cellulose and lignocellulosic biomass in an aqueous solution.
- 3.3 million atoms
- Reaction-field electrostatics instead of PME

3.2.4.4 NAMD

NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems. Based on Charm++ parallel objects, NAMD scales to hundreds of cores for typical simulations and beyond 500,000 cores for the largest simulations.

Test Dataset used

20stmv2fs.namd: This input file comes from the official NAMD benchmark dataset provided by the University of Illinois at Urbana-Champaign. It was chosen because it scales efficiently on large HPC systems while simulating a moderately large, heterogeneous protein–solvent system.

NAMD dataset

Dataset characteristics:

- Represents a 5×2×2 replication of the original STMV dataset from the official NAMD site.
- Contains approximately 20 million atoms.

3.2.4.5 QuantumESPRESSO

Quantum ESPRESSO is an integrated suite of Open-Source computer codes for electronic-structure calculations and materials modelling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.

Test Dataset used

Ta2O5-2x2xz-552.in: This input file is also extracted from the [Unified European Applications Benchmark Suite (UEABS)] QuantumESPRESSO dataset. It was selected as it is large, computationally intensive, and stresses both CPU/GPU computation and inter-node communication, while being representative of real quantum materials simulations.

Dataset characteristics:

- Simulates Tantalum oxide (TA2O5) with a supercell (2x2x2)
- Represents a real scientific case not just a synthetic input.

3.2.4.6 WRF

WRF, or the Weather Research and Forecasting model, is a state-of-the-art numerical weather prediction system for both atmospheric research and operational forecasting.

Test Dataset used

IP-4KM: This input is a real operational Iberian Peninsula model with a resolution of 4 km, with a grid size of 400×400×38. The Simulation covers 54 forecast hours, and it is designed to evaluate real climate weather forecast scenario checking the code **scalability and performance** of WRF for operational workloads on HPC systems.

Dataset characteristics:

- Real dataset input provided by the BSC Earth science team.
- 4km resolution for a forecast simulation of 54 hours.
- Performance and scalability limited sometimes for the I/O part of the code, included in the benchmark.

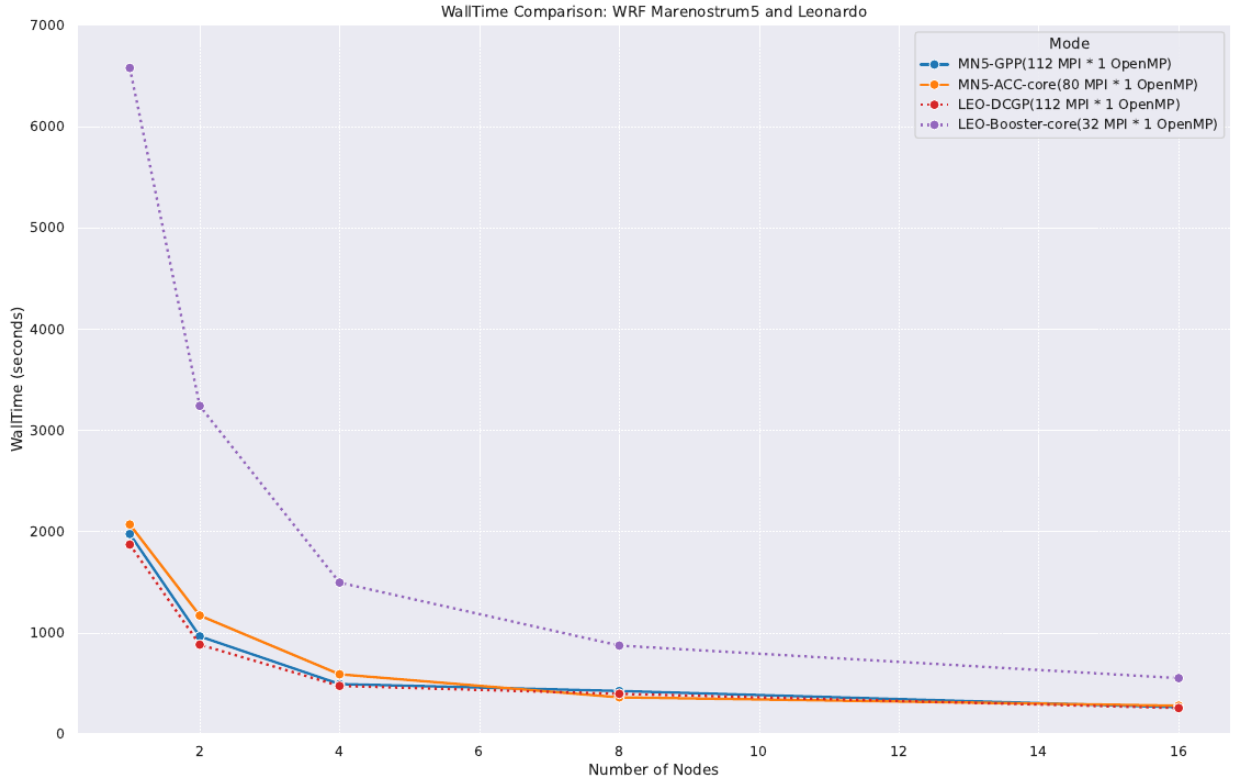
3.2.5 Results

The EPICURE Benchmark Suite has been tested on two EuroHPC JU systems up to now, but it should be extended to other during the lifetime of the project:

- MareNostrum 5[6] (Spain): A pre-exascale supercomputer with both CPU and GPU partitions, used to assess the suite's performance on different hardware architectures.
- LEONARDO[7] (Italy): Another pre-exascale system with diverse partitions for data-centric and GPU-focused tasks, providing a varied testing environment.

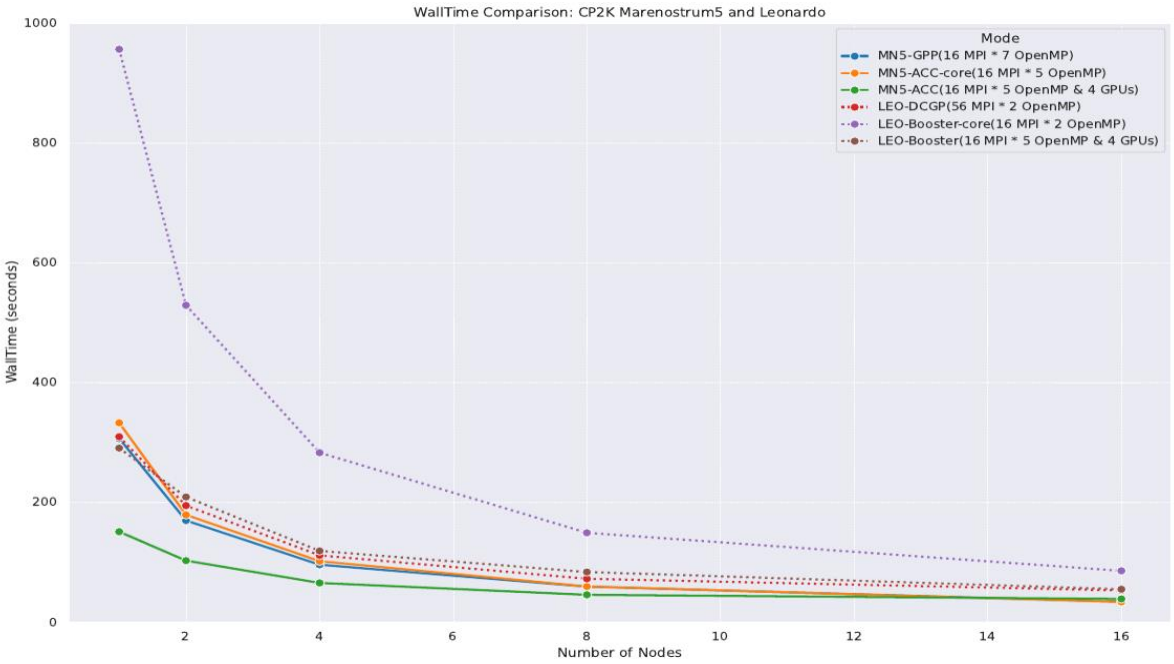
Initial results from benchmarking and performance analysis show that the suite successfully automates the installation and execution of scientific applications, with the following key outcomes:

- Scalability Insights: The system provides detailed information on the scalability of applications, including the impact of single node versus multi-node configurations.
- Architecture/machine comparison: The system provides detailed information about the execution of the same application on different systems/partitions. Showing the best hardware/site per application/input.
- User-Friendly Automation: EPICURE ASTs have been able to use the suite with minimal efforts, demonstrating its effectiveness in simplifying complex workflows.



2 – Figure with the WRF results from the benchmark suite running on MN5 and Leonardo

[6] MareNostrum5 supercomputer, https://www.bsc.es/es/marenostrum/marenostrum-5
[7] Leonardo Supercomputer, https://leonardo-supercomputer.cineca.eu



3 – Figure with the CP2K results from the benchmark suite running on MN5 and Leonardo

Overall, the EPICURE Benchmark Suite has proven to be an effective tool for automating software installation, benchmarking, and performance analysis in HPC systems. The tool will enable the EPICURE team to detect possible cross-site optimization, architecture efficiency for applications and detect difference between sites at configuration level, which can improve or decrease the performance of specific applications.

3.3 Performance analysis

In this section, we present some of the performance analysis conducted during the first 12 months of the EPICURE project. The projects highlighted here represent only a selection of the analyses and profiling efforts completed as part of this task, but they provide a good overview of the type of work undertaken.

The work has been divided into two categories:

- Profiling Work: This category focuses on identifying the most critical parts of the code that require further optimization efforts and CUDA kernels analysis.
- Full Application Analysis: This category involves evaluating the entire application to identify its strengths and weaknesses. The goal is to provide developers with insights and recommendations to improve the scalability and overall performance of their applications in a comprehensive setup.

3.3.1 PROFILE WORK

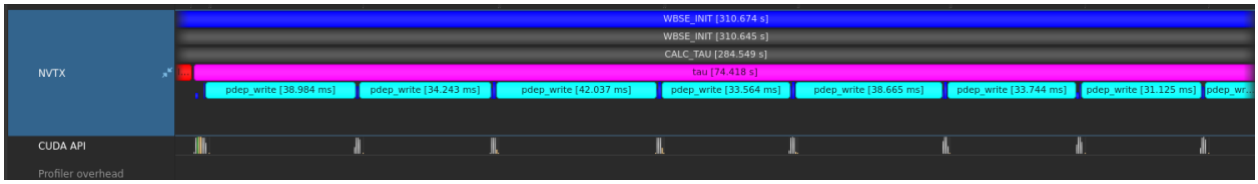
This section highlights the types of profiling work conducted on the task. The selected examples include the profiling of an MPI[8]+OpenACC[9]+CUDA[10] code with I/O routines, an AI Python code, and an MPI+OpenACC Fortran code. These examples effectively demonstrate the scope of work carried out during this period and the variety of projects received.

3.3.1.1 Project #2: Profiling and IO Optimization Analysis of WEST Quantum Chemistry Code

The WEST (Wide-scale Electronic Structure Toolkit) quantum chemistry package performs many-body perturbation theory calculations using a hybrid MPI+OpenACC/CUDA implementation in Fortran. The code makes extensive use of IO operations for storing intermediate calculation results, particularly during the initialization phase (wbse_init.x).

The analysis utilized Nvidia Nsight Systems[11] with NVTX markers strategically placed to isolate performance-critical regions. The profiling revealed that IO operations constituted a significant performance bottleneck, with synchronous write operations blocking computational kernels.

Initial profiling highlighted that write operations in wbse_init.x exhibited poor overlap with computation, leading to CPU stalls. The profile data as seen by in Figure 4 shows sequential patterns where compute kernels would wait for IO completion before proceeding.



4 – Figure of Nsight Systems trace showing sequential execution of computation and IO

We implemented asynchronous IO operations by creating a background IO Thread Pool, this part of work as implied code modifications, was done inside work package 3, so we will not describe it here.

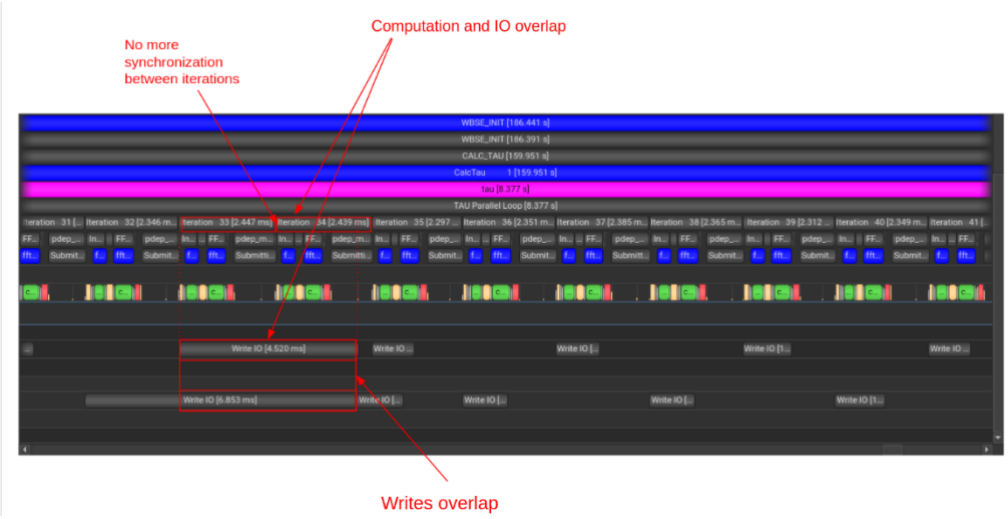
After the modifications done in WP3, we were able to verify the successful overlap of computation with IO writes in the background thanks to the Nvidia Nsight trace.

[8] MPI standard, https://en.wikipedia.org/wiki/Message_Passing_Interface

[9] OpenACC programming model, https://en.wikipedia.org/wiki/OpenACC

[10] CUDA programming model, https://en.wikipedia.org/wiki/CUDA

[11] Nvidia Nsight profiler and tracer system from NVIDIA, https://developer.nvidia.com/nsight-systems



5 – Figure of Nsight Systems trace showing successful overlap of computation and IO.

3.3.1.2 Project #9: Profiling Analysis of AtmoRep: A Python Machine Learning Code Optimization

AtmoRep is a stochastic model for atmospheric dynamics using transformer-based deep learning, implemented in Python with PyTorch. The codebase processes large-scale atmospheric data using NumPy and Zarr for data handling, with a neural network containing 3.5 billion parameters. The primary optimization target was the data processing pipeline, specifically the MultifieldDataSampler class responsible for loading and preprocessing atmospheric state data.

A multi-tool profiling approach was necessary to gain comprehensive insights into different aspects of the code performance. The following tools were employed:

1.cProfile[12] (Python built-in profiler)

- Primary tool for overall program flow analysis
- Limited to profiling the main thread only
- Successfully identified that 89.62% of total execution time was in the iter function

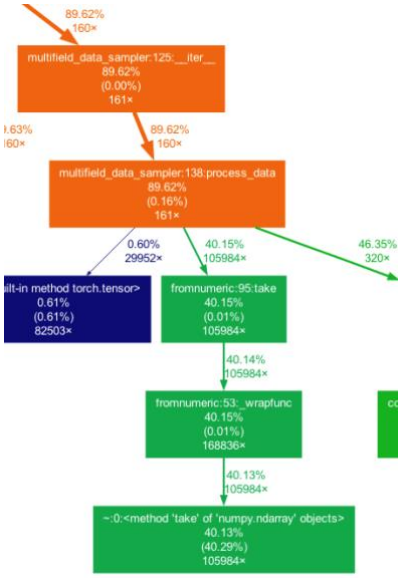
2.Line Profiler[13]

- Provided line-by-line execution timing

[12] cProfile python profiler, https://docs.python.org/3/library/profile.html#module-cProfile

[13] Line profiler for Python, https://pypi.org/project/line-profiler/

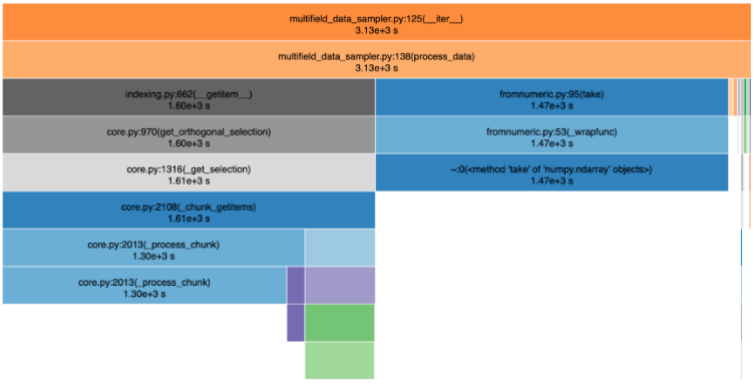
- Revealed critical bottleneck: *data_tt=self.ds['data'].oindex[idxs_t]* consumed 1868 seconds out of 2133.29 seconds total



6 – Figure of GraphViz visualization profile for project #9

3.Visualization Tools

- **SnakeViz**[14]: Provided interactive visualization of **cProfile** data as seen in Figure 6
- **GraphViz**[15]: Enabled identification of performance hotspots through visual representation as seen in Figure 7



7 – Figure of SnakeViz showing execution bottlenecks for project #9

[14] Snakeviz visualizer, https://jiffyclub.github.io/snakeviz/

[15] GraphViz for python, https://pypi.org/project/graphviz/

The profiling effort revealed several critical insights about code performance and tool effectiveness:

- NumPy array operations in data loading were primary bottlenecks
- Multiple unnecessary array copies in data processing using numpy's fancy indexing
- Inefficient use of NumPy's take() function was detected.

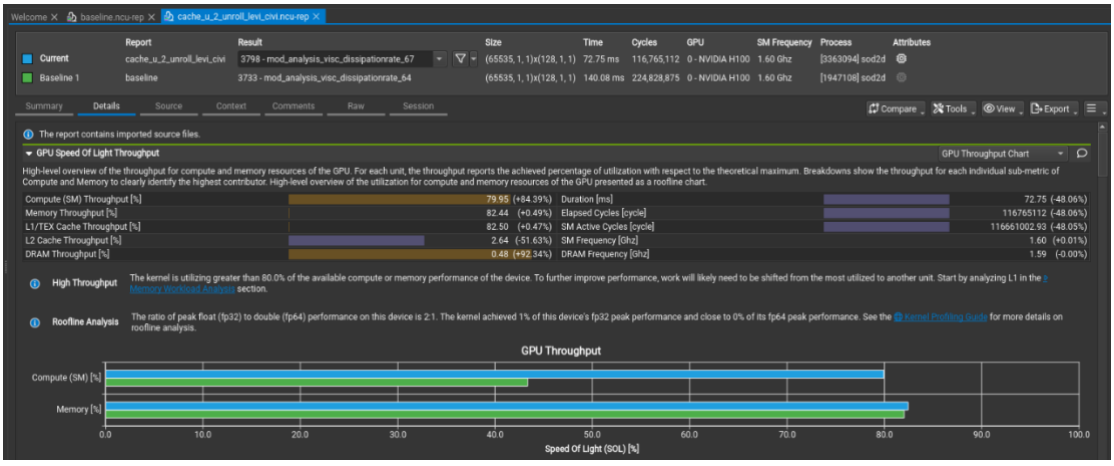
3.3.1.3 Project #16: Profiling Analysis of GPU-Accelerated CFD Code Using NVIDIA Nsight Compute

The target application is a large-scale computational fluid dynamics (CFD) code implemented in Fortran with GPU acceleration via OpenACC directives. The code is part of the DEAREL framework that combines CFD simulations with deep reinforcement learning for active flow control. The primary objective was to utilize NVIDIA Nsight Compute to identify performance bottlenecks and optimization opportunities in the GPU-accelerated kernels.

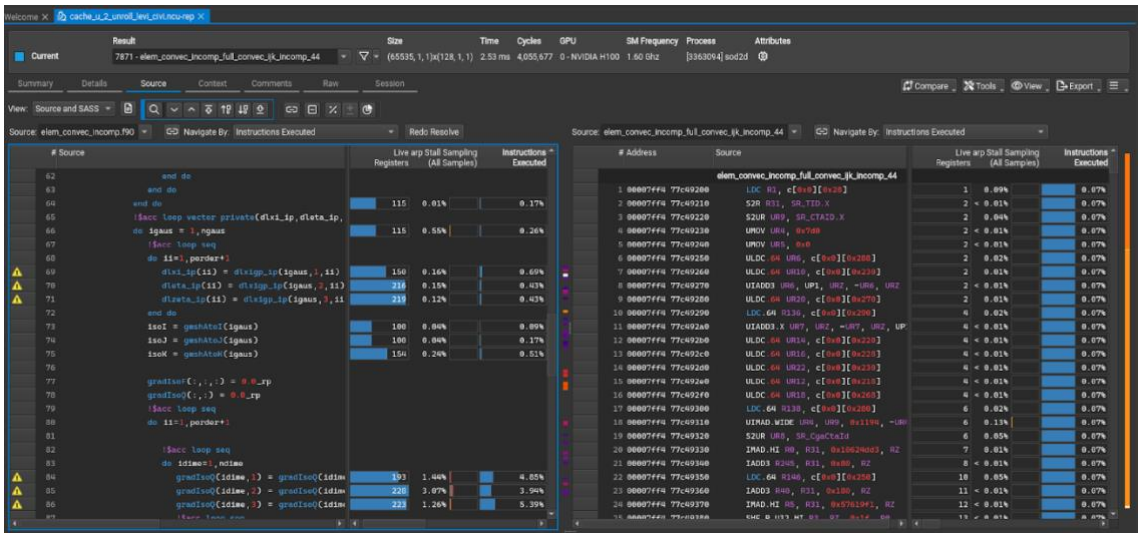
NVIDIA Nsight Compute was employed to collect detailed performance metrics and analyse kernel execution characteristics. The profiling focused on:

1. GPU compute and memory throughput metrics as seen in Figure 8
2. Memory access pattern analysis
3. Kernel occupancy and resource utilization
4. Source-level correlation of performance bottlenecks as seen in Figure 9

Key kernels were profiled individually to obtain detailed performance metrics. The source-level analysis capability of Nsight Compute was particularly valuable in identifying specific code regions responsible for suboptimal performance.



8 – Figure Showing GPU utilization metrics



9 – Figure Showing GPU source correlation of performance bottlenecks

Nsight Compute profiling revealed three primary performance bottlenecks:

- Uncoalesced memory access patterns
- Low kernel occupancy due to register pressure
- Suboptimal loop structures limiting parallelism.

3.3.2 FULL application analysis

In this section, a few examples of comprehensive application performance analysis conducted during the period are highlighted, offering insights into the type of analysis that can be performed for applications under EPICURE support.

3.3.2.1 Project #3: Anthropogenic Stressors and Future Changes of the Northwest European Shelf ecosystem

The primary objective of the project was to determine the optimal configuration - compiler, compiler flags and MPI implementation - that would give the best performance on the Vega[16] system. In addition, it was desired to profile the application to identify bottlenecks and explore potential performance improvements. The IZUM team focused on identifying the optimal compiler flags and MPI implementation, while FZJ was responsible for collecting profiles and traces, followed by detailed performance analysis.

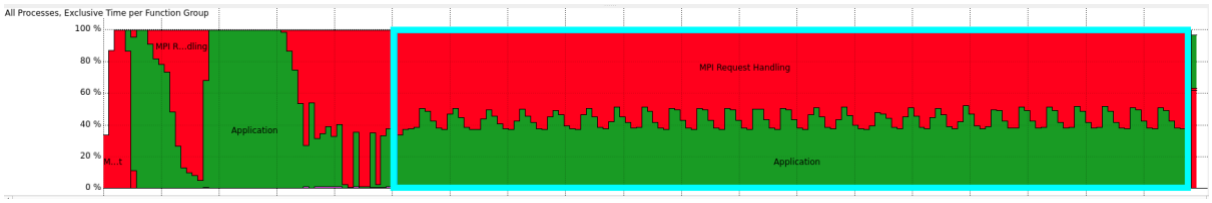
Figure 10 illustrates the summary timeline of the total application execution on three nodes of the Vega CPU partition (658 MPI ranks). The data was collected using Score-P17 and visualized with

[16] Vega supercomputer, https://izum.si/en/vega-en/

[17] Scalable Performance Measurement Infrastructure for Parallel Codes, https://score-p.org

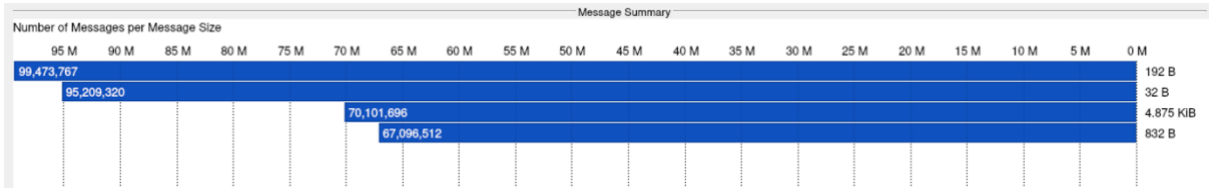
Vampir18. This chart depicts the percentage of processes engaged in specific tasks over time. Green represents computation, while red indicates MPI routines. The application's structure follows the typical pattern of scientific applications: initialisation, a solver loop, and finalisation. The solver loop, highlighted by the blue frame, is the largest and typically the most computationally intensive phase, making it the primary focus of our analysis (FOA).

A closer examination of the solver loop reveals that a significant portion of the execution time is spent in communication, predominantly in *MPI_Waitall* operations. These operations indicate time spent waiting for non-blocking communications to complete. Consequently, our subsequent analysis concentrated on optimising communication patterns to mitigate this bottleneck.



10 – Figure of a summary timeline of the application collected with Score-P on Vega. Executed on 3 nodes (658 MPI ranks)

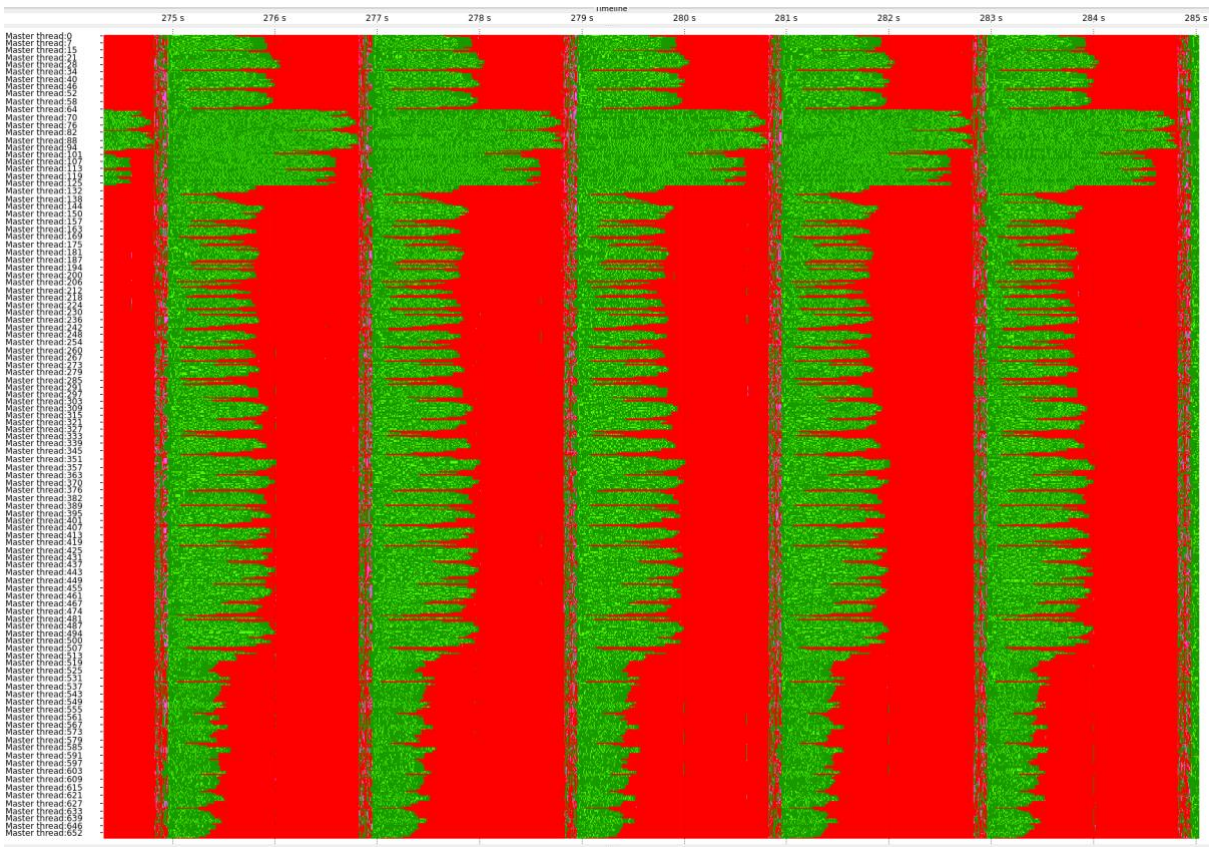
The communication analysis revealed that the application communicates by sending a large number of very small messages, ranging in size from 32 bytes to 4.9 KiB (see Figure 11). Additionally, the analysis showed that the number of messages varies across MPI ranks, with ranks assigned to higher-numbered processes sending and receiving slightly more messages than those with lower numbers.



11 – Figure of the number of messages per message size for the total solver loop.

A closer look at five iterations of the FOA (see Figure 12) discovered a significant imbalance in the workload across MPI ranks. This imbalance is likely due to the input files' structure or shape, resulting in an uneven distribution of computational and communication tasks.

[18] Event trace visualizer, https://vampir.eu



12 – Figure of a Zoom-in into five iterations of the solver loop.

To improve performance, it is recommended that the identified imbalances be addressed and the communication strategy optimized. Specifically:

- **Load balancing:** evenly distribute workloads among MPI ranks. This can be achieved by assigning more processes to regions with higher workloads or by using dynamic load-balancing libraries such as DLB[19], Charm++[20].
- **Communication optimization:** reduce the overhead associated with small message sizes by aggregating data into larger messages whenever possible. This approach minimizes communication overhead and improves bandwidth utilization.

It is also recommended to encourage the development team to participate in tuning workshops and hackathons organized by the EuroHPC JU and national HPC centres.

[19] Dynamic Load Balancing Library: https://www.bsc.es/research-and-development/software-and-apps/software-list/dlb-library-dynamic-load-balancing

[20] Charm++ parallel programming framework, https://charmplusplus.org

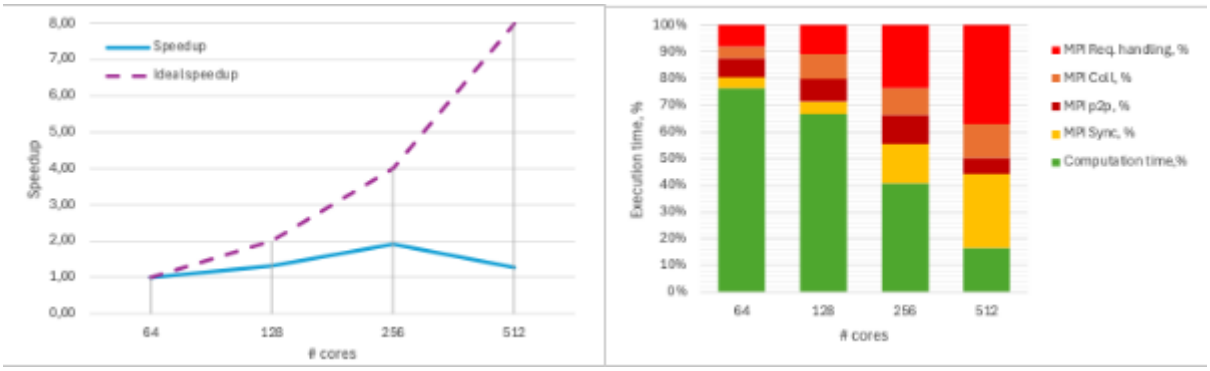
3.3.2.2 Project #11: Large-Eddy Simulations of Supersonic Air Intakes

The project aimed at identifying why performance degrades (2-3x) as core usage increases within a single node, propose solutions, and implement them, focusing on data structures and compute-intensive routines. The LuxProvide team focused on memory layout and vectorization of a specific routine, while FZJ looked at internode behaviour.

This is a typical scientific application including initialisation, solver loop and finalisation. The focus in the current work was on the solver loop (FOA). Scaling tests were performed on the MeluXina[21] CPU partition using a 64x64x64 testcase with 10 iterations, running on 1 to 4 nodes. Measurements for 64-core and 128-core configurations were executed on a single node.

As shown in Figure 13 (left), the application's speedup is far from ideal, with efficiency decreasing significantly from 1.0 at 64 cores to 0.16 at 512 cores. The execution breakdown in Figure 13 (right) indicates that computation time decreases while MPI time, mostly for request handling and synchronisation, increases.

The provided testcase is too small to demonstrate effective strong scaling. As a next step, larger test cases were explored to address this limitation.

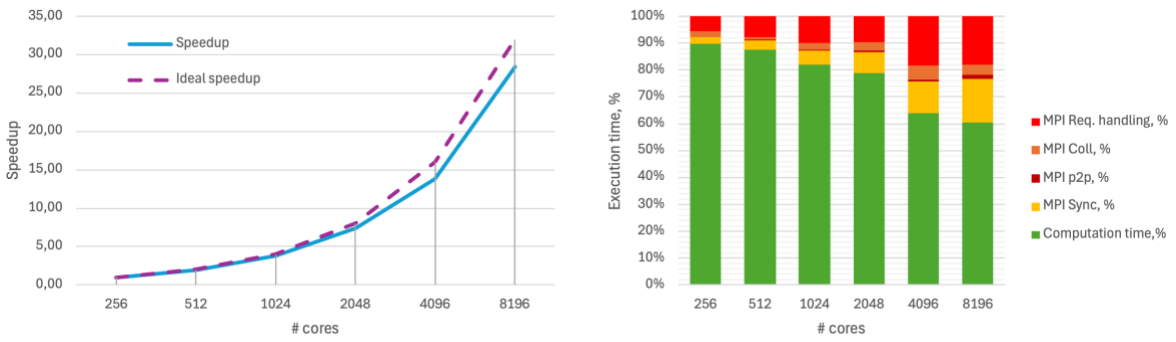


13 – Figure of the scaling plot of FOA, 10 iterations, 64x64x64 testcase (left), breakdown of execution time (right)

To better utilize the available computing resources, the input testcase was increased by a factor of 512. Scaling tests were executed on the MeluXina CPU partition using a 512x512x512 test case across 2 to 64 nodes, with 10 iterations per configuration. The tests began with 2 nodes, as the problem could not fit into a single node.

[21] MeluXina system overview, https://docs.lxp.lu/system/overview/

Figure 14 (left) shows that the application achieved near-ideal speedup, with efficiency decreasing from 1.0 (256 cores) to 0.89 (8196 cores). The execution breakdown (Figure 14, right) indicates that computation time decreases with scaling, while MPI overhead, primarily from request handling and synchronization, grows, but much more gradually compared to smaller testcases.



14 – Figure of scaling plot of FOA, 10 iterations, 512x512x512 testcase (left), breakdown of execution time (right)

The application's execution at scale was assessed using the methodology and metrics developed by the POP CoE[22]. Table 15 lists the computed POP metrics[23] for the large test case. These metrics are expressed as efficiencies ranging from 0 to 1, with higher values indicating better performance. According to POP methodology, efficiencies above 0.8 are generally considered acceptable, while lower values signal performance issues that require further investigation.

As shown in Table 15, *Global efficiency* remains acceptable up to and including 2048 cores. At larger scales, *Global efficiency* metrics show a slight decline, primarily due to reduced *Serialisation efficiency*. This metric reflects dependencies across processes, leading to alternating processes waiting for each other. To pinpoint the issue, Scalasca[24] trace analysis was used to identify the specific inter-process dependencies causing communication-related waiting times.

[22] POP CoE, https://pop-coe.eu/

[23] POP metrics, https://pop-coe.eu/node/69

[24] Scalasca (Scalable Performance Analysis of Large-Scale Applications, https://www.scalasca.org/scalasca/software/scalasca-2.x

POP efficiency metrics	# cores					
	256	512	1024	2048	4096	8196
Global Efficiency	0,90	0,89	0,84	0,82	0,71	0,70
- Parallel Efficiency	0,90	0,87	0,82	0,75	0,59	0,53
-- Load Balance Efficiency	0,96	0,95	0,94	0,92	0,90	0,90
-- Communication Efficiency	0,94	0,92	0,87	0,82	0,66	0,59
--- Serialisation Efficiency	0,95	0,94	0,89	0,85	0,70	0,62
--- Transfer Efficiency	0,99	0,98	0,98	0,97	0,94	0,95
- Computation Efficiency	1,00	1,01	1,02	1,10	1,21	1,31
-- Instruction Scaling	1,00	1,00	0,99	0,98	0,97	0,96
-- IPC Scaling	1,00	1,02	1,03	1,12	1,25	1,37
Computation time	14881,55	14636,03	14306,98	13433,19	12106,25	10944,02
IPC	0,60	0,61	0,62	0,67	0,75	0,82
Instructions (only computation)	2,80E+13	2,81E+13	2,83E+13	2,85E+13	2,89E+13	2,93E+13
Data cache misses	2,63E+11	3,04E+11	2,95E+11	2,65E+11	2,32E+11	2,01E+11

15 – Table of POP efficiencies of FOA, 10 iterations, 512x512x512

A Scalasca trace analysis identified several inefficient communication patterns (see Table 16), with two of them, “Wait at Barrier” and “Late Sender Wrong Order”, showing significant growth at scale. The analysis pinpointed the source code locations and highlighted potential causes, such as redundant *MPI_Barrier* routines and frequent file existence checks. These findings were shared with the project team. A detailed discussion of the source code is omitted here due to the proprietary nature of the software.

Communicattion pattern (% of total runtime)	# cores					
	256	512	1024	2048	4096	8196
Wait at Barrier, %	2,29	3,17	5,15	9,14	15,71	20,58
Barrier Completion, %	0	0,14	0,05	0,05	0,98	0,76
Late Sender ,%	0,1	0,23	0,48	1,17	2,63	3,8
Late Sender Wrong Order,%	0,39	1,01	2,12	3,43	9,59	10,56
Late Reciever, %	4,43	5,42	6,04	5,77	5,19	3,86
Late Broadcast, %	0,7	0,18	0,54	0,79	0,81	1,83
Wait an NxN, %	0,99	0,68	1,72	0,71	1,27	2,35

16 – Table of summary of identified inefficient communication patterns of FOA, 10 iterations, 512x512x512

During the project, it was found that the small problem (64x64x64) was not very appropriate at scale, while the larger case (512x512x512) had an acceptable efficiency, i.e. 0.89 (at 8196 cores). Further investigation using the POP methodology identified serialisation inefficiencies at larger scale. Findings such as "Wait at Barrier" and "Late Sender Wrong Order" provide clear directions for future optimisation.

4. T2.3 - Best Practices Guidelines for EuroHPC supercomputers and technical reports

4.1 Introduction and status

During the first 12 months of the project, efforts have been concentrated on defining potential topics for the 2025 user guidelines and identifying projects that could serve as the foundation for technical reports to be published in the second project year.

A technical meeting held in December 2024 in Brussels provided valuable insights into the current topics most likely to be developed into technical reports. This meeting also offered useful information for refining or expanding the Best Practices guidelines. The meeting included all technical teams from EPICURE and featured project-specific presentations to showcase the current status, progress, and key insights. This collaborative setting enabled teams to share their work, exchange ideas, and gain support on common topics.

4.2 Best Practices guidelines

The list of possible topics selected in the first months of the project is still valid, but some improvements have been added following the expertise gained during the first 12 M.

The initial list generated and already presented in the M6 review was the following:

- Power Consumption Measurements in EuroHPC JU systems
  - A guide detailing the various methods available on EuroHPC JU systems to extract power consumption data per job/node, accompanied by examples and real job scripts for testing.
  - General recommendations for optimizing energy efficiency in jobs (e.g., best energy configurations for tasks).
  - Exploration of other related topics on power consumption.
- Optimization of AI Workflows for EuroHPC JU systems
  - Best practices for running AI workflows across different EuroHPC JU systems architectures (AMD, NVIDIA, etc.).
  - Strategies for optimizing parallel AI workflows and addressing common challenges.
  - Additional topics such as installation, execution, job configuration, and I/O considerations for AI workflows.

- Potential Topics for Future Guidelines

- Modern and emerging architectures (e.g., exascale systems, future processors).
- Programming methods for accelerated systems, including current techniques and their benefits/drawbacks, with specific recommendations for future code development.
- Comprehensive workflows: covering the full process from transferring data and code to obtaining results on the machine.
- Handling system restrictions on large European machines, such as lack of outgoing network connections, and their implications for software installation or development workflows.
- Strategies for addressing the "lots of small files" issue, including containerization of Python and Conda installations and code adaptation strategies.

Currently, the primary focus is on the first topic, Power Consumption Measurements in EuroHPC JU systems, which was established in Month 6 (M6) as the project's initial guideline due Month 18 (M18). During our state-of-the-art analysis, we identified a similar guideline, “Harvesting Energy Consumption on European HPC Systems: Sharing Experience,” developed by the CEEC project[25] and published on 12 August 2024, which is available here.

While the CEEC document covers some of the content initially planned for our guideline, which was already in preparation when the CEEC guideline was published in August, it also references potential future work by CASTIEL in this area. After consulting with CASTIEL representatives, we confirmed that they do not plan to produce a broader guide on this subject within their scope, thus leaving room for us to adapt our initial guideline.

Based on the available information, the focus of our guidelines has shifted to a more hands-on, practical approach. It will incorporate real examples that users can test on various machines, with actual outputs from each system. This approach ensures that our document complements the CEEC guideline, which is more theoretical, and avoids redundancy.

The First Best Practice Guideline is schedule to be released and publicly available in M18.

[25] CEEC COE for Fluid Dynamics, https://ceec-coe.eu/

5. Conclusions

The first 12 months of the project required coordination both at the project level among teams and with EuroHPC JU to assign projects. This initial phase resulted in reduced optimization and enabling activities due to the limited number of assigned projects. Consequently, resource utilization was lower during the first six months.

This issue was addressed through improved allocation processes with EuroHPC JU and enhanced internal collaborations, enabling multi-site support for projects where multiple areas of expertise were required. These efforts improved collaboration between sites and brought new projects under EPICURE's scope. Thanks to these adjustments, WP2 utilization improved during the second half of the year. However, additional projects are still needed to achieve a more balanced workload distribution.

Despite these challenges, WP2 has successfully achieved the initial objectives of the work package, delivering significant improvements and optimizations for the users and projects of EuroHPC JU resources. These efforts have included crucial optimizations specifically aimed at simplifying and enhancing the utilization of the EuroHPC JU machines for the selected projects.