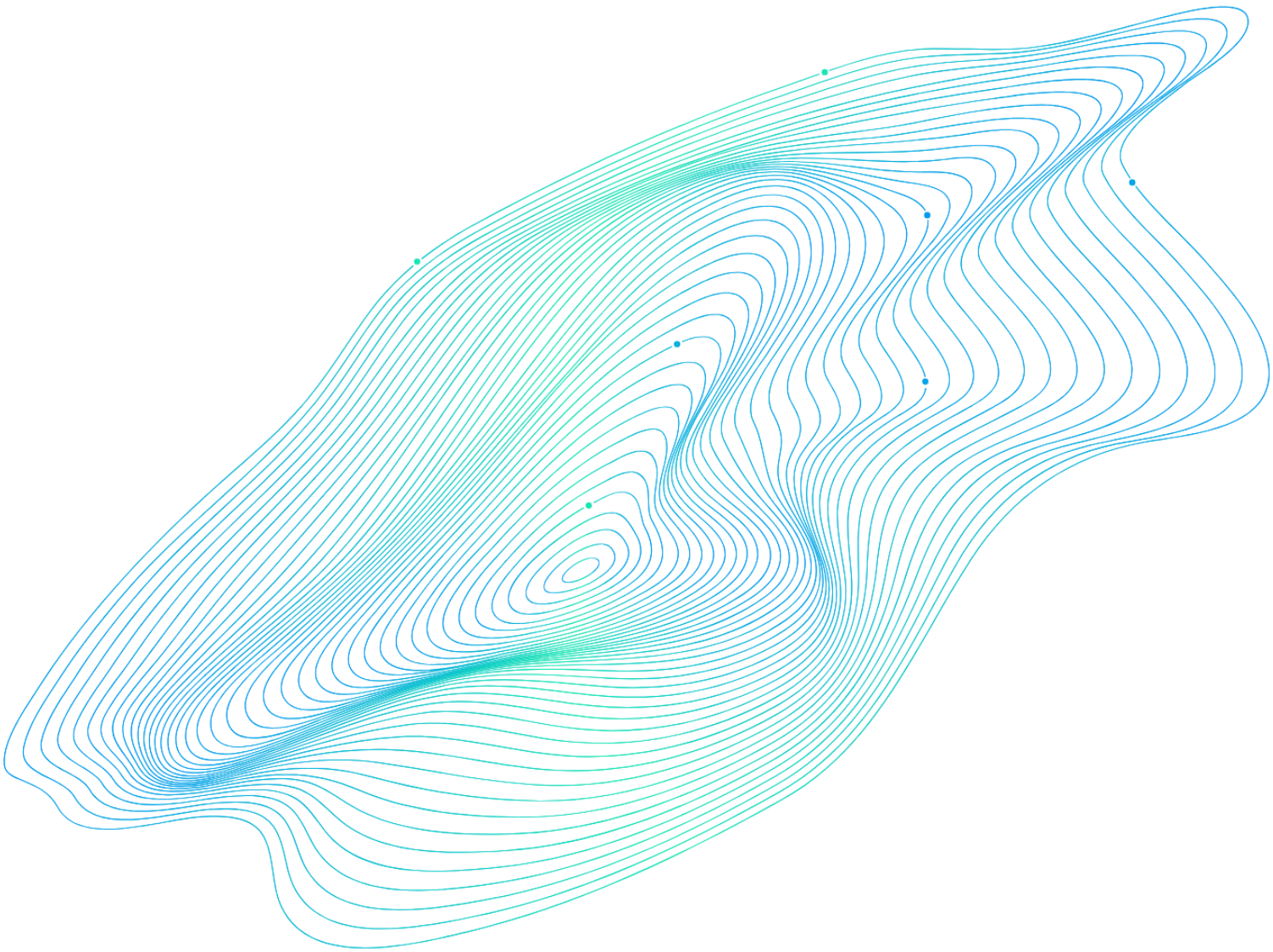
EPICURE

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 2.2

Report on code porting, enabling and scaling for Year 2



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) Federico Padron (BSC-CNS)
Contributors	Ziyad Albanoby (BSC-CNS) Gaurav Saxena (BSC-CNS) ALL EPICURE WP2 team
Peer Reviewers	Stefan Becuwe (UAntwerpen) Julius Roeder (DEIC)
Version	1.0
Due Date	31/01/2026
Submission Date	29/01/2026

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	10/12/2025	David Vicente (BSC)	Table of Contents
0.2	02/01/2026	David Vicente (BSC)	First complete draft of the deliverable
0.3	06/01/2026	Stefan Becuwe (UAntwerpen) Julius Roeder (DEIC)	Review conducted by the appointed reviewers
0.4	19/01/2026	David Vicente (BSC)	Reviewers input consolidated and second draft sent to the Project Management Board (PMB) for Final comments
0.5	21/01/2026	Ville Savolainen (CSC) Joao Barbosa (IT4I)	PMB feedback on the deliverable received
0.6	22/01/2026	David Vicente (BSC)	PMB feedback consolidated and final version of the deliverable made available to the PMO
1.0	29/01/2026	Virvatuli Uusimäki (CSC) Alberto Lanzanova (CSC)	Final quality check performed by the Project Management Office and submission of the deliverable

Declaration on the use of AI

This report has been prepared with the support of AI tools which were used to assist in drafting text and checking language. All content has been thoroughly reviewed, fact-checked, and edited by the authors to ensure accuracy and alignment with the objectives of the report.

Glossary of Terms

Item	Description
ACC	Accelerated Partition (referring to MareNostrum 5 Accelerated – ACC – partition)
CoE	Centre Of Excellence
EAR	Energy Aware Runtime software
EuroHPC-JU	The EuroHPC Joint Undertaking, a public-private partnership responsible for developing and deploying high-performance computing (HPC) infrastructure across Europe
FEM	Finite Elements Method
GPP	General Purpose Partition (referring to MareNostrum 5 General Purpose – GPP – partition)
HPC	High Performance Computing
MPI	Message Passing interface
OpenMP	OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared-memory multiprocessing programming models
WP	Work Package

Executive Summary

During the second year, EPICURE achieved significant progress in optimizing multiple projects under Work Package 2 (WP2). The focus has been on providing Level 2 support, concentrating on code compilation optimization, workflow efficiency, code analysis, profiling, benchmarking, and job distribution set-up.

At the site level, partners collaborated effectively, leveraging their combined expertise within WP2 to deliver specialized support to projects. These efforts resulted in measurable improvements in the performance and efficiency of up to 57 projects, facilitating more effective use of EuroHPC resources.

Table of Contents

1. Introduction 8
2. Level 2 support to EuroHPC projects 9
2.1 Introduction and status 9
2.2 Detailed description per project 13
3. EPICURE Benchmarking 47
3.1 Introduction 47
3.2 Design 47
3.3 Implementation for M12 to M24 47
3.4 Future work 50
4. Best Practices Guidelines for EuroHPC supercomputers and technical reports 51
4.1 Introduction and status 51
4.2 Best Practices Guidelines 51
5. Conclusions 53

List of Tables and Figures

1 - Table listing of projects and their optimization type of work for WP2 or WP3. 13
2 - Table with the comparison between ARM and X86 for Gromacs 18
3 - Graph comparing Leonardo and Google Collab A100 GPUs 20
4 - Table showing the improvement on time thanks to EPICURE support for project 139 21
5 - Figure showing the speed-up provided when adding Pararicks. 23
6 - Figure showing the speed-up of using ELPA instead of ScaLAPACK. 31
7 - Figure with the speed-up difference between GPU and CPU version of the code 41
8 - Figure with the metrics generated from JUBE for CP2K on 1 node. 48
9 - Figure with CP2K results from the benchmark suite running on different supercomputers. 49
10 - Figure with CP2K wall time efficiency on different supercomputers. 49

1. Introduction

The second year of EPICURE focused on supporting the projects assigned by EuroHPC JU. Each project was managed by a lead partner who coordinated with the principal investigators to define the expected work and assign the appropriate collaborators from the EPICURE Applications Support Teams (AST).

Thanks to process improvements in project requests and enhanced coordination with EuroHPC JU, the number of supported projects at WP2 level, increased significantly, from approximately 30 projects in the first year to 57 projects in the second year.

This deliverable is structured to present the work carried out across three main areas:

- Level 2 support for EuroHPC projects: Tasks include code porting, project enabling, performance analysis, and performance improvements.
- EPICURE benchmarking activities: Covering all work related to the EPICURE benchmarking framework, the application benchmark suite, and their installation and set-up across various EuroHPC supercomputers.
- Best Practices Guidelines and technical reports for EuroHPC supercomputers: Including the work in the first Best Practice Guideline published at M18 and its update at M24.

2. Level 2 support to EuroHPC projects

2.1 Introduction and status

During the second year, EPICURE supported 41 projects under WP2 only, 18 projects through a combination of WP2 and WP3, and 5 projects under WP3 only. The projects listed here are those that received some level of optimization during the second year and were not already reported in Deliverable D2.1. Projects 50 and 57 were included in D2.1 for their WP2 activities; however, they are also listed in the table below because additional effort was carried out under WP3 during this reporting period. Consequently, these two projects do not appear in the detailed WP2 section, which includes only the 57 projects supported at the WP2 level that were not previously reported. Projects limited to an initial meeting or brief exchange, without follow-up optimization work, are excluded, as the associated effort was minimal.

In the list below, each project is categorized based on the type of work provided: WP2 (Level 2 support) or WP3 (Level 3 support). While WP2 activities are detailed in this deliverable, WP3 work is documented in D3.2.

ID	Type of call	Grant Number	Title	Lead Partner	Status	WP2	WP3
164	Development	EHPC-DEV-2025D12-072	Energetic particle physics in tokamak plasmas using hybrid approach with full-orbit and full-F simulations	BSC	In Progress	X	
162	Benchmark	EHPC-BEN-2025B11-069	Scalability and Performance Monitoring for Multilingual Continued Pre-Training of European LLMs	BSC	In Progress	X	
160	Development	EHPC-DEV-2025D11-141	Development of a Single-Cell-Centered Foundation Model for Digital Pathology	IZUM	In Progress	X	
159	Regular Access	EHPC-REG-2024R02-191	High-fidelity scale resolving simulation of the flow around a rounded step (HIFRS)	IZUM	In Progress	X	
156	Development	EHPC-DEV-2025D08-106	Scalability in FPGA clusters with EPSILOD: A framework for generic iterative loop stencil applications	LUXPROVIDE SA	In Progress		X
150	Benchmark	EHPC-BEN-2025B10-036	Benchmarking Dedalus	CSC	In Progress	X	
149	Benchmark	EHPC-BEN-2025B10-004	Tinker-HP benchmarking on AMD GPUs	DeiC (DTU)	In Progress	X	
146	Benchmark	EHPC-BEN-2025B06-034	Towards a high-resolution data-driven weather forecasting system over France	CINES	Completed	X	
145	Regular Access	EHPC-REG-2025R01-105	Decoding Sec14-like SFH Protein Disorder and Promiscuity via Molecular Dynamics SECure (ehpc335)	BSC	In Progress	X	

143	Benchmark	EHPC-BEN-2025B08-024	Understanding the binding to CXCR4 of the anti-HIV-1 peptide pepRF1 using computational tools	INESC TEC	Completed	X	
142	Benchmark	EHPC-BEN-2025B09-026	Towards Sustainable Ammonia Synthesis: DFT-Guided Design of Molecular Electrides and Metalloaromatic Clusters	UAntwerpen	Completed	X	
140	Development	EHPC-DEV-2025D08-047	Using machine-learning guided binder discovery for advancing pathology biomarker research	IZUM	In Progress	X	X
139	Regular Access	EHPC-REG-2025R01-091	High resolution regional historical and future simulations of climate and atmospheric composition	BSC	In Progress	X	
137	Development	EHPC-DEV-2025D07-067	Distributed LLMs on ARM	INESC TEC	In Progress	X	X
135	Development	EHPC-DEV-2025D04-052	DT Laser Cut	SofiaTech	In Progress	X	X
133	Development	EHPC-DEV-2025D02-008	Scaling Deep Neural Networks for Public Safety: Fire, Flood etc. with HPC	BSC	Completed	X	X
132	Benchmark	EHPC-BEN-2025B07-009	Exploring additional computational resources for Latvian genome reference project	LUXPROVIDE SA	Completed	X	
130	AI	EHPC-AI-2024A06-006	European Regulatory Compliance Copilot: An AI-Powered Solution for Real-Time Compliance Monitoring and Reporting	BSC	In Progress	X	
129	AI	EHPC-AI-2024A06-008	Insoore GenAI - Transforming Insurance Claims Processing Through GenAI-driven Damage Assessment and Fraud Prevention	CINECA	On Hold by PI request	X	
128	Development	EHPC-DEV-2025D05-006	Past Stall Operation of Hydrogen Multistage Radial Compressors	CINECA	In Progress	X	X
127	Regular Access	EHPC-REG-2024R02-151	Seareport	LUXPROVIDE SA	Completed	X	
126	Benchmark	EHPC-BEN-2025B05-003	Event Building at the Future Circular Collider: Machine Learning Based Particle Flow and Track Reconstruction	BSC	Completed	X	X
125	Development	EHPC-DEV-2025D02-093	Development of the gPLUTO code to AMD GPU with OpenMP	CSC	In Progress	X	
123	Development	EHPC-DEV-2025D01-005	Intense ultra-short laser-matter interaction: applications to material modification and high-harmonic generation in solids	SofiaTech	In Progress	X	X

122	Development	EHPC-DEV-2025D04-139	Developing a Heart Language Foundation Model	LUXPROVIDE SA	Completed	X	
121	Development	EHPC-DEV-2025D04-116	CCTV Surveillance Video Understanding	IT4I@VSB	Completed	X	
120	Development	EHPC-DEV-2025D04-122	Normalisation-Independent Differential Expression Using the Propr R package within NF-Core Differentialabundance	BSC	In Progress		X
118	Development	EHPC-DEV-2025D05-093	More extreme scalability in EPSILOD: A framework for generic iterative loop stencil applications (2)	UAntwerpen	Completed	X	
116	Benchmark	EHPC-BEN-2025B03-064	LLM-Powered Malware Analysis: Detection, Localization, and Privacy Risks in Android Applications	IZUM	Archived	X	
115	Development	EHPC-DEV-2025D03-071	Porting EMTO on GPU	KTH	In Progress	X	
114	Development	EHPC-DEV-2025D04-097	Large-scale fission properties with energy density functionals	UAntwerpen	In Progress	X	
113	Development	EHPC-DEV-2025D02-094	Compilation and porting of the code THATMPI using Cray compiler in LUMI	CYFRONET	Completed	X	X
112	Benchmark	EHPC-BEN-2025B04-049	Wasco AI – Custom AI Assistant Optimization via HPC	CINECA	Completed	X	
111	Benchmark	EHPC-BEN-2025B02-048	Scaling memory use in Yambo code for spectroscopic applications	CYFRONET	Completed	X	
110	AI	EHPC-AI-2024A02-044	Pretraining General Document Understanding Models	BSC	Wait for Feedback	X	X
109	Development	EHPC-DEV-2025D04-113	Exploiting GPU computational power to simulate a turbulent flow over a grid spaced 127 pin bundle using SPUMA	CINECA	In Progress	X	X
108	Development	EHPC-DEV-2025D04-035	GPU-enabling of explicit compressible OpenFOAM solvers for flow and acoustic prediction	CINECA	In Progress		X
107	Development	EHPC-DEV-2025D04-021	BCPNNSim	CINES	Completed	X	X
106	Development	EHPC-DEV-2025D04-077	Enhancing EUBERT: Retraining the European Parliament's LLM with Contemporary Parliamentary Data	LUXPROVIDE SA	Completed	X	
105	Development	EHPC-DEV-2024D12-048	NowcastAI HPC	CINECA	On Hold by PI request	X	

104	Development	EHPC-DEV-2025D01-065	HPCW development	BSC	In Progress	X	
103	Regular Access	EHPC-REG-2024R02-178	CANONICS: ChArge aNd current ON cosmIC Strings	CINECA	In Progress	X	
102	Regular Access	EHPC-REG-2024R02-131	CyberShake physics-based simulations for machine learning ground motion forecasting in South Iceland	BSC	In Progress	X	
99	Regular Access	EHPC-REG-2024R02-140	Monte Carlo simulations of spin glasses for quantum-simulation experiments	CSC	In Progress	X	X
98	Development	EHPC-DEV-2024D11-038	Molecular dynamics of TPPS4 aggregates	IT4I@VSB	Completed	X	
97	Benchmark	EHPC-BEN-2024B11-044	Non-orthogonal configuration interaction for electron and energy transfer	BSC	Completed	X	
96	Development	EHPC-DEV-2025D12-017	Horses3d in many GPUs	BSC	Wait for Feedback	X	X
95	AI	AI-BOOST	Calico	CYFRONET	In Progress	X	
94	Development	EHPC-DEV-2024D12-094	GPU Optimization and Scalability of the Poisson Solver in BigDFT for Advanced Electronic Structure Calculations on LUMI	CINES	In Progress	X	
92	Development	EHPC-DEV-2024D05-021	LIG: Exploring Geoparsing and Large Language Models in German Abstractive Summarization Tasks	IT4I@VSB	Completed	X	
90	AI	EHPC-AI-2024A04-095	Revolutionizing Industrial Enzymes through Generative AI	CINECA	Completed	X	
89	Development	EHPC-DEV-2024D11-037	Predictive Exploration of Correlation in Specific Interactions and Optimization Networks (PRECISION)	CINECA	Completed	X	X
88	AI	EHPC-AI-2024A04-062	Geo-Llama, the first LLM for geographic data	IT4I@VSB	In Progress	X	
86	AI	EHPC-AI-2024A04-084	On Device AI Assistant: Revolutionizing Real-time API Interactions with Privacy Preserving Intelligence	BSC	In Progress	X	
85	Development	EHPC-DEV-2024D11-072	Scalable and Intelligent Traffic Modeling for Improved Urban Planning	IT4I@VSB	In Progress	X	X
79	Development	EHPC-DEV-2024D09-067	Energy Efficiency in Industry 5.0: A Genetic Algorithm for Energy Consumption Optimization	LUXPROVIDE SA	Wait for Feedback	X	

78	Development	EHPC-DEV-2024D09-054	POP CoE 2024	INESC TEC	Completed	X	
70	Benchmark	EHPC-BEN-2024B09-047	Financial Virtual Assisstant for Portfolio Management (FVAP)	LUXPROVIDE SA	Wait for Feedback	X	
62	Development	EHPC-DEV-2024D10-054	A high-order efficient and scalable solver for NWP using dynamically adaptive non-conforming meshes	LUXPROVIDE SA	Wait for Feedback	X	X
57	Extreme	EHPC-EXT-2024E01-064	Quantum Monte Carlo generation of machine learning potentials for hydrogen-based high-temperature superconductors	CINECA	Completed	X	X
50	Development	EHPC-DEV-2024D07-076	Machine Learning of Exchange Energy Density and Integration with Python-based Electronic Structure Codes	LUXPROVIDE SA	Completed	X	X
48	Benchmark	EHPC-BEN-2024B07-050	Cross-Facility Federated Learning	LUXPROVIDE SA	In Progress		X
34	AI	EHPC-AI-2024A02-043	Unsupervised artificial intelligence models for Down syndrome neuroimaging research: from automatic brain alteration detection to the generation of synthetic 3D magnetic resonance images	BSC	Wait for Feedback	X	X
32	AI	EHPC-AI-2024A02-037	CAPHARD: Codon-Assisted Phage-Host Automatic inteRaction Discovery	CINECA	Completed		X
ID	**Type of call**	**Grant Number**	**Title**	**Lead Partner**	**Status**	**WP2**	**WP3**

1 - Table listing of projects and their optimization type of work for WP2 or WP3.

2.2 Detailed description per project

This expanded version provides a detailed account of the activities and outcomes of each project, including for each project, a brief description highlighting its focus, key activities undertaken, the resulting outcomes and a conclusion.

The order followed in the list is from newer to older (as older projects have lower project IDs from EPICURE).

Project 164	**EHPC-DEV-2025D12-072**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	The project aimed to analyse and address performance bottlenecks in the XTAPAS simulation code and identify the root causes of performance degradation and memory access inefficiencies.	

Work done	Initial profiling with Nsight Systems revealed that nearly all runtime was consumed by the boris_push_particles kernel, which also showed high standard deviation in execution time. Low GPU occupancy suggested memory access issues. Further analysis of file write events and iteration timing indicated that the gap between iterations grew over time, not due to I/O but likely because of changes in kernel behaviour. The team post-processed profiling traces to visualize iteration times and kernel call frequencies, confirming that the boris_push_particles kernel's runtime increased significantly as the simulation advanced. To isolate the issue, the code was modified to sample the kernel at regular intervals using Nsight Compute. Profiling results showed a dramatic rise in global memory traffic (1573.84% increase from first to last iteration) and reduced cache hit rates, pointing to poorly coalesced or irregular memory access patterns. Long scoreboard stalls in later iterations further indicated increased data processing demands.
Improvements provided	The analysis pinpointed the primary bottleneck: escalating global memory reads from GPU HBM. The team demonstrated that file writes were not the cause of slowdowns, but rather the kernel's growing memory access inefficiencies. These insights provide a clear direction for optimization in the code for future WP3 work.
Conclusions	The investigation successfully identified the boris_push_particles kernel as the main performance bottleneck, with memory access inefficiencies worsening over time and across different memory architectures. The findings highlight the importance of hardware-aware optimization and suggest that future work should focus on memory access patterns and data management to mitigate performance degradation in long-running simulations.

Project 162	**EHPC-BEN-2025B11-069**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	The project aimed to assist users in setting up and optimizing their environment in MareNostrum 5 (MN5) for the execution of LLM training workloads. The focus was on adapting Slurm job scripts and Singularity images, originally developed for the Leonardo supercomputer, to function correctly on MN5.	
Work done	Support was provided primarily through email - addressing user questions about job queue submission, storage access and performance, and the downloading & accessing of AI models. The team helped users transition from Leonardo's modules (openmpi/4.1.6-nvhpc-24.3, gcc/12.2.0, nvhpc/24.3) to their MN5 equivalents and resolved initial issues with pip installations by configuring the Python environment properly.	
Improvements provided	We reduced the time to set up the user's workflow on the machine, enabling a better use of their computational time.	
Conclusions	The project successfully facilitated the adaptation of the user's environment to MN5, allowing their LLM training workloads to run seamlessly & efficiently.	

Project 160	**EHPC-DEV-2025D11-141**	IZUM
Machines: Vega		Collaborators: None
Project goals	The project aimed to test and optimize a PyTorch-based foundation model deployment for the Vega Supercomputer, ensuring efficient training and	

	scalability across multiple GPUs and nodes.
Work done	The team developed training scripts, added performance tracing, and implemented RAM and processor monitoring. The final phase focused on scaling the scripts for multi-node use. We benchmarked the training for the model on 2, 4 and 6 nodes (8/16/24 GPUs), with an almost perfect scalability • 2 nodes (8 GPUs): 320 s • 4 nodes (16 GPUs): 130 s • 6 nodes (24 GPUs): 75 s
Improvements provided	We provided the researchers with guidance, sample scripts, and performance monitoring tools for their training. We found a particularly good scalability when going from 2 to 6 nodes, which enables faster training.
Conclusions	The project successfully optimized the code for Vega, addressing technical challenges and delivering a robust, scalable solution.

Project 159	**EHPC-REG-2024R02-191**	IZUM
Machines: Vega		Collaborators: None
Project goals	Users faced issues with repeated job failures when doing large-scale runs (256 nodes) using PETSc.	
Work done	We initially provided support to the user on how to configure the MPI rank distribution across nodes and how to clear cache as an unprivileged user. After the first benchmark input was provided, we identified errors related to PETSc linking. PETSc was then recompiled using OpenMPI and the compilation completed successfully. However, during the execution phase, additional errors appeared in the user's code, which are still under investigation.	
Improvements provided	Recompilation of PETSc using OpenMPI and some user questions resolved in the initial stage of the project.	
Conclusions	The project is still in progress. Significant advances have been made in the compilation stage, but persistent issues when running the user code still not solved.	

Project 157	**EHPC-DEV-2025D10-057**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	The goal was to enable a set-up around vLLM (virtual Large Language Models) that was flexible enough for the researchers but could work under the network constraints of MareNostrum 5 (MN5) (no outbound connection from compute nodes).	
Work done	The team initially explored the use of a Singularity image containing all necessary packages. However, this approach proved limiting as it hindered interoperability with the researchers' established Python workflows. As an alternative, a Conda environment was created, tailored to the specific modules required by the researchers. Additionally, scripts and detailed guidance were provided to support the offline use of Weights & Biases (WanDB) for monitoring ML training.	
Improvements provided	Enabled the researcher's workflow under MN5's constraints.	

Conclusions	The project successfully addressed the challenge of running vLLM-based workflow on MN5 without the need for outbound network access and provided documentation for the user on how to manage WanDB connection despite the MN5 network limitation.

Project 150	**EHPC-BEN-2025B10-036**	CSC
Machines: LUMI		Collaborators: IT4I@VSB
Project goals	The project aimed to build and deliver production-ready containers for both v2 and v3 of the Daedalus framework.	
Work done	Containers for Daedalus v2 and v3 were successfully built and delivered using a strategy adapted from a previous project, with guidance from ARCHER2's build instructions for (similar) Cray systems. While the process required some adjustments, the PI received the containers and completed the required benchmarks.	
Improvements provided	Provided containers with Daedalus v2 and v3.	
Conclusions	We were able to create the containers for Daedalus v2 and v3 allowing them to run on LUMI. The researchers informed us that this also fixed an issue with HDF5 present in other Cray installations.	

Project 149	**EHPC-BEN-2025B10-004**	DeiC (DTU)
Machines: LUMI		Collaborators: None
Project goals	The project focused on installing and running Tinker-HP (Fortran + OpenMP + HIP) and Deep-HP (Python + JAX + CuPy + FeNNol) efficiently on MI250x GPUs on LUMI, while ensuring compatibility between the CCE compiler, ROCm, JAX, CuPy, and all tools required for production runs. An additional objective emphasized by the project leader was to address performance issues, particularly the 3–5x slowdown observed in the Fortran/OpenMP AMD GPU code compared to NVIDIA A100 GPUs.	
Work done	The initial work involved building containerized JAX and ROCm environments using Velocity to ensure reproducible deployments, including MPI and RCCL stacks. JAX 0.7.1 on ROCm 7.0.2 and JAX 0.6.0 on ROCm 6.2.4 (with necessary patches) were successfully built. FeNNol was integrated, tested, and ReFrame benchmarks were created, while the JAX profiler was instrumented. Profiling was attempted with Omnitrace, rocprof-sys, TAU, and Nsight; Omnitrace traces remained incomplete. Containers were also prepared to be compatible with the upcoming LUMI software stack upgrade.	
Improvements provided	• More reproducible testing and deployment via container workflows. • Better understanding of ROCm/JAX compatibility boundaries and required patches. • Initial performance benchmarking infrastructure via ReFrame. • Working profiling pipeline using TAU and JAX tools where Omnitrace tracing failed.	
Conclusions	JAX set-up on MI250x requires careful dependency control and version matching and for profiling Omnitrace currently produces incomplete traces for JAX workloads. TAU profiling works more reliably for GPU traces.	

	The project is technically prepared for the LUMI upgrade while further optimization and profiling work continues.

Project 146	**EHPC-BEN-2025B06-034**	CINES
Machines: LUMI		Collaborators: IT4I@VSB
Project goals	The project aimed to diagnose and improve the scalability and stability of distributed deep learning training workloads using the Anemoi framework on the LUMI supercomputer. As part of a broader benchmark campaign, the goal was to identify the root causes of non-deterministic communication failures, especially when scaling beyond 16–32 nodes and to distinguish between software, hardware, and configuration-related issues affecting performance on heterogeneous GPU-based European HPC systems.	
Work done	The team conducted extensive benchmarking and diagnostic efforts on LUMI, which is equipped with AMD MI250x GPUs and the Cray Slingshot interconnect. Observations revealed non-systematic crashes and timeouts during inter-node communication, with failures becoming more frequent as the number of nodes or model input size increased. Smaller runs (≤4 nodes) remained stable, suggesting interconnect saturation or misconfiguration as likely culprits. Several potential causes were investigated, including hardware instability, software stack inconsistencies (Anemoi, PyTorch Lightning, ROCm, RCCL), RCCL/NCCL communication layer issues, and improper interconnect configuration. Profiling attempts, such as enabling detailed logging, often altered runtime behaviour, indicating sensitivity to timing and environment variables. Diagnostic approaches included inserting barriers between collective primitives, profiling runs, and using the RCCL Replayer to reproduce communication traces offline. The team confirmed the functionality of the CXI library and proposed single-rank-per-node experiments to isolate network-level causes. Containerized and bare-metal deployments were compared, with the latter recommended for better control and reduced interference.	
Improvements provided	The investigation narrowed down the likely sources of instability to the communication stack, particularly the interaction between RCCL, libfabric (CXI), and the Cray interconnect. The team identified that the use of the CXI plugin was critical for performance and stability, and that fallback to TCP caused significant degradation.	
Conclusions	While the exact root cause of the communication instability on LUMI remains undetermined, the project made significant progress in isolating the conditions under which failures occur. The findings suggest that the issues are closely tied to the communication stack and network configuration.	

Project 145	**EHPC-REG-2025R01-105**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	Porting and installation of OpenMM (https://openmm.org/) via EasyBuild.	
Work done	We installed and created a module under the MareNostrum 5 (MN5) GPP partition (OpenMM/8.3.1-foss-2025a). The researchers successfully ran their simulations with it. The requested help from the PI was very limited so no more efforts were required.	

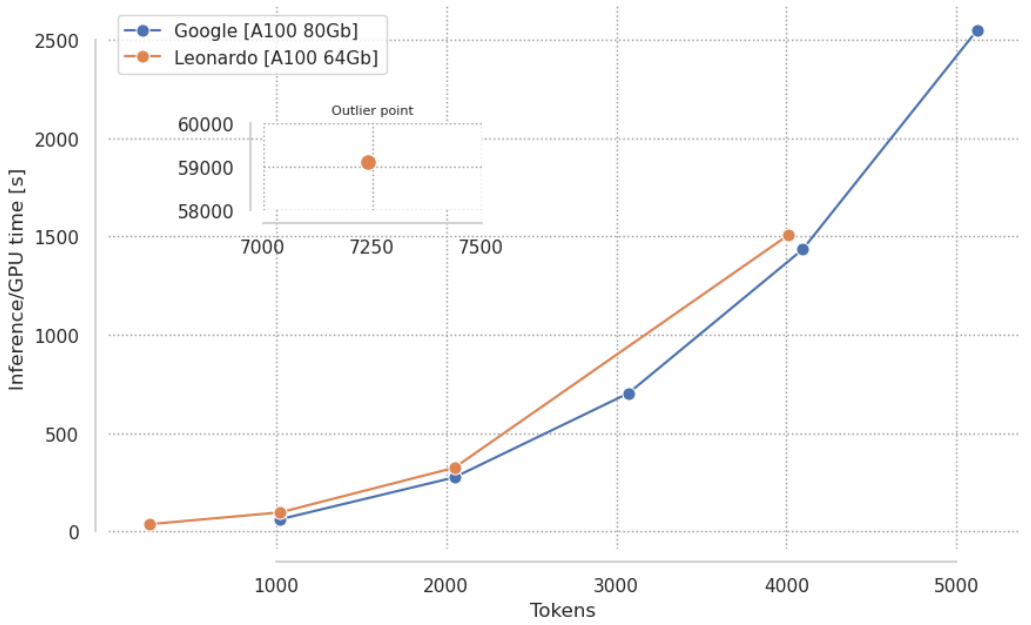
Improvements provided	Provided OpenMM module within EasyBuild.
Conclusions	OpenMM was successfully ported to and installed on MN5 using EasyBuild.

Project 143	**EHPC-BEN-2025B08-024**	INESC TEC
Machines: Deucalion		Collaborators: None
Project goals	Benchmarking and optimization of GROMACS performance on the x86 and ARM CPU partitions of Deucalion, aiming to identify and resolve performance bottlenecks encountered during production simulations of a ~130k-atom system.	
Work done	We provide extended documentation on this project at: https://opencode.it4i.eu/epicure/project-143-ehpc-ben-2025b08-024 We did some scalability tests in x86 and ARM, with the following results: Nodes | X86 perf | x86 Config MPIxOMP | ARM perf | ARM Config MPIxOMP | Speed up per Node | Speed up per Core 1 | 59.549 | 64 × 2 | 15.981 | 48 × 1 | 3.73 | 1.39 2 | 111.490 | 32 × 4 | 28.796 | 48 × 1 | 3.87 | 1.46 4 | 198.871 | 32 × 4 | 51.703 | 12 × 4 | 3.84 | 1.44 *2 - Table with the comparison between ARM and X86 for Gromacs* x86 achieves higher performance than ARM across all tested nodes. On a single node, x86 reaches 59.55 ns/day (64 × 2) compared to 15.98 ns/day (48 × 1) for ARM, with a speedup of ~3.7× per node and ~1.4× per core. On 2 and 4 nodes, x86 achieves 111.49 ns/day and 198.87 ns/day, while ARM reaches 28.80 ns/day and 51.70 ns/day, corresponding to speedups of ~3.8× per node and ~1.45× per core.	
Improvements provided	From this project, we can extract new guidelines for GROMACS usage on Deucalion: 1) Optimal efficiency is obtained for 1-2 x86 nodes. 2) ARM nodes are convenient for medium to small systems when using more than 2 nodes. 3) x86 remains faster than ARM nodes for high-throughput calculations.	
Conclusions	An interesting result from this exploration is that PME 3-D FFT/MPI overheads dominate when using more than 2 nodes. Overall, GROMACS usage seems to perform better under x86 nodes, and for small systems, the scalability saturates quickly.	

Project 142	**EHPC-BEN-2025B09-026**	UAntwerpen
Machines: LUMI, Leonardo		Collaborators: None
Project goals	Install Gaussian, ORCA, CP2K or OpenMolcas, and benchmarking them.	

Work done	Most questions were related to general HPC usage and login issues. Support for Leonardo was limited to providing the user with general advice for using Gaussian16 on the machine. For LUMI, installation instructions were provided for CP2K, which are publicly available (https://lumi-supercomputer.github.io/LUMI-EasyBuild-docs/c/CP2K/). For ORCA, a version of OpenMPI was required to run the program. We installed OpenMPI, then ORCA, and managed to get a stable run on a single node. We also installed OpenMolcas with all the dependencies and linked to the Cray-libsci.
Improvements provided	We provided a new, fine-tuned OpenMPI installation on LUMI to work with the specific cray hardware. We tested ORCA on LUMI and found a ~2x speed increase on a small program when using local /dev/shm or /tmp instead of shared/scratch (or similar).
Conclusions	Most of the issues faced in this project were addressed speedily, thanks to the existing HPC documentation. ORCA provided an interesting case to test the importance of the file system used in computations. In addition, the results of this project has already proved useful for users beyond EPICURE (e.g. with the new ORCA installation on LUMI).

Project 140	**EHPC-DEV-2025D08-047**	IZUM
Machines: Leonardo, Vega		Collaborators: CINECA
Project goals	The project aimed to deploy, optimize, and benchmark a suite of bioinformatics and computational tools—including AlphaFold3, Chai, Boltz1, Boltz2, NanoMelt, AbNativ, TNP, and AbPro—on the HPC Vega and Leonardo systems.	
Work done	The project commenced with the installation and testing of Chai, Boltz1, and Boltz2 on HPC Vega, followed by the acquisition of AlphaFold3 model parameters from Google. Instructions for installing these models were documented and shared with the principal investigator (PI). Benchmarking of AlphaFold3 on Vega revealed performance bottlenecks, particularly in the HMMER suite, which was later recompiled with MPI support to improve scalability. Several software tools (NanoMelt, AbNativ, TNP, and AbPro) were successfully installed on Vega as requested. GPU and CPU monitoring scripts were developed to track resource usage, generating visualizations for analysis. The team also created a Singularity image for AbPro and conducted extensive benchmarking of Chai and Boltz2 under various Slurm configurations. Collaborative meetings with the PI and the CINECA team were held to discuss progress, particularly focusing on the 15k production run for Boltz1 and AlphaFold3 optimizations. A markdown guide was prepared to document installation and benchmarking procedures on Leonardo, including AlphaFold3 performance comparisons between Leonardo's A100 64GB GPUs and reference A100 80GB GPUs (https://hackmd.io/@lsalicar/ryetq53tlx).	

	 *3 - Graph comparing Leonardo and Google Collab A100 GPUs*
Improvements provided	The project delivered optimized configurations for AlphaFold3, identifying the HMMER suite as a critical bottleneck and implementing MPI support to enhance its performance. Benchmarking revealed that using one GPU per prediction, with tasks-per-node=4 and cpus-per-task=2, provided the best balance for HMMER's multithreading capabilities. With this set-up, we achieved the same results with Leonardo's GPU as those that Google obtained with more memory per GPU.
Conclusions	The project successfully installed, tested, and optimized a range of computational tools on HPC Vega and Leonardo, with a focus on AlphaFold3 and Boltz1. Key bottlenecks were identified and addressed, particularly in the HMMER suite, leading to improved scalability and performance. The developed monitoring tools and benchmarking procedures will facilitate future optimizations and large-scale runs. Collaboration with CINECA was crucial to avoid repeated efforts.

Project 139	**EHPC-REG-2025R01-091**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	Compilation and profiling of a given WRF (Weather Research and Forecasting) model input (https://www.mmm.ucar.edu/models/wrf) provided by the user.	
Work done	We created a Conda environment named climate-tools in General purpose partition (GPP) to be used by researchers for their simulations. The main approach for this work involved targeting the I/O-related options, since I/O was diagnosed as the bottleneck for the input provided. We tested the code in 1, 2 and 3 nodes, with the optimal time achieved for 2 nodes. Using 3 nodes instead of 2 only provided a minor speed-up.	
Improvements provided	We provided the installation and gave the users the optimal configuration for their inputs, thus saving energy, time and resources. The improvements were the followings thanks to the specific I/O configuration used for the WRF input execution:	

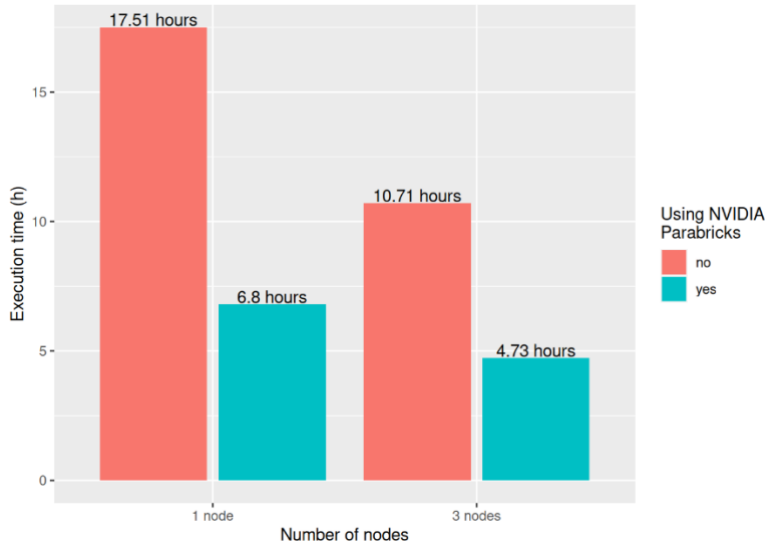
	<table><tr><th>Nodes</th><th>User Best Walltime</th><th>EPICURE Optimized Walltime</th><th>Improvement</th></tr><tr><td>1</td><td>01:07:22</td><td>00:49:18</td><td>~27% faster</td></tr><tr><td>2</td><td>00:47:06</td><td>00:30:30</td><td>~35% faster</td></tr><tr><td>3</td><td>00:39:49</td><td>00:25:07</td><td>~37% faster</td></tr></table> *4 - Table showing the improvement on time thanks to EPICURE support for project 139*
Conclusions	We successfully provided the WRF installation and provided a better WRF configuration (using specific I/O processes for checkpointing and output generation) for the user's input which worked up to 37% faster than the previous user runs with the same number of nodes.

Project 137	**EHPC-DEV-2025D07-067**	INESC TEC
Machines: Deucalion		Collaborators: None
Project goals	Get a working vLLM set-up on Deucalion CPU nodes.	
Work done	Installed vLLM on Deucalion CPU nodes. Installed and configured Ray for distributed execution. Verified dependencies, Python environment, and system libraries on ARM nodes.	
Improvements provided	Provided users with a vLLM set-up with Ray, reducing their set-up time.	
Conclusions	Successfully installed a vLLM set-up with Ray on Deucalion ARM nodes.	

Project 135	**EHPC-DEV-2025D04-052**	SofiaTech
Machines: Discoverer		Collaborators: None
Project goals	The goal was to resolve the compilation of the MOOSE solver on Discoverer. The solver codebase is old and effectively unsupported and hence this required adapting it to the software stack on Discoverer.	
Work done	We successfully compiled and installed MOOSE, paying attention to potential performance improvements. Both manual build procedures and the provided automated installation recipe were analysed and adapted to the Discoverer software stack. The build was carried out using the GCC toolchain with OpenMPI and integrated with the HDF5 and NetCDF libraries supplied through the system module environment.	
Improvements provided	A complete and reproducible installation recipe was developed, resulting in a working and optimized MOOSE installation tailored to Discoverer.	
Conclusions	The work demonstrates that, despite its age, MOOSE can be reliably deployed on modern HPC systems when its dependencies and configuration are carefully managed.	

Project 133	**EHPC-DEV-2025D02-008**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	The goal was to provide guidance for efficient deep learning training on MareNostrum 5, covering profiling & debugging, energy & carbon footprint estimation, suitable hardware selection, data handling practices, storage & quota scalability, and the feasibility & efficiency of multi-node multi-GPU execution for YOLOv8.	
Work done	We delivered a Slurm submission script enabling profiling using NVIDIA Nsight Systems and allowing generation of profiling reports for offline analysis. Guidance was provided on extracting job energy consumption using the EAR accounting module, along with instructions for checking project quotas and resource usage. We also recommended data pre-processing strategies to optimize I/O performance. Finally, we modified the YOLOv8 codebase (part of WP3 deliverable), Slurm scripts and the user source code to enable multi-node multi-GPU execution and evaluated its scaling behaviour.	
Improvements provided	The profiling workflow now enables detailed performance and memory analysis, while energy usage can be directly retrieved at the job level. Clear recommendations were established for hardware selection and data handling, thus improving training efficiency. Resource usage analysis showed low quota consumption and good single-node and multi-node scaling for standard training, confirming that a 10× dataset increase is feasible within the current allocation.	
Conclusions	We enabled YOLOv8 usage for large datasets. Single-node multi-GPU training now scales efficiently.	

Project 132	**EHPC-BEN-2025B07-009**	LUXPROVIDE SA
Machines: MeluXina		Collaborators: None
Project goals	The objective was to assess and optimize the performance of a genomic analysis workflow based on the Sarek pipeline on the MeluXina supercomputer, with a focus on improving execution time through I/O tuning, multi-node scalability, and GPU acceleration.	
Work done	We first ported and executed the benchmark workflow on the MeluXina CPU partition and performed an initial performance analysis using Nextflow's built-in reporting tools. This profiling identified the GATK4_MARKDUPLICATES tasks as the main bottleneck due to long run-times, low CPU utilization, and heavy I/O. Several I/O configurations were tested using Tier-2 storage, Tier-1 storage, and RAM disk, followed by experiments with multi-node execution. To efficiently exploit multiple nodes, Nextflow was reconfigured to use the HyperQueue executor. Finally, the workflow was ported to GPU nodes and updated to support NVIDIA Parabricks within recent Sarek versions, resolving missing reference data and compatibility issues.	
Improvements provided	The main improvement was achieving a 3.1x speed-up with respect to the original configuration, greatly accelerating the user's workflow. In the graph below we show part of the optimization that was the GPU porting using Parabricks and was one of the main speed-up improvements of the project.	

	 *5 - Figure showing the speed-up provided when adding Parabricks.*
Conclusions	Our optimization provided a significant speed-up (3.1x times). Most of the speed-up stems from algorithmic changes enabled by NVIDIA Parabricks, with a contribution from HyperQueue providing scalable task distribution for larger workloads. Storage and memory optimizations only provided marginal gains.

Project 130	**EHPC-AI-2024A06-006**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	Help users set up their DeepSpeed-ACCELERATE framework within MareNostrum 5 (MN5).	
Work done	We created an example for ACCELERATE with DeepSpeed optimizations and shared it with the users. We also helped them set up their required datasets and provided an example of WandB offline usage in MN5 for training monitoring. Finally, we created a Conda environment for the users with all their required packages.	
Improvements provided	By creating the environment and providing examples, we effectively circumvented all potential issues related to set-up, enabling researchers to focus exclusively on running their AI training.	
Conclusions	We provided all the packages and software stack required by the researchers. In addition, we now have an DeepSpeed-ACCELERATE example, including scripts, modules and a job submission example, that can be provided to other users requiring similar setup on MN5.	

Project 129	**EHPC-AI-2024A06-008**	CINECA
Machines: Leonardo		Collaborators: INESC TEC
Project goals	Provide guidelines on AI training in CINECA.	
Work done	We held two different meetings with the users to guide them on how to use Leonardo's modules containing optimized libraries and software, the cineca-ai module and to build a Python virtual environment on top of it. Further, we provided tips on how to use it for AI fine-tuning.	

Improvements provided	We helped researchers set up their simulations by providing insight on Leonardo modules and Slurm allocation guidelines.
Conclusions	After both meetings, the researchers had a clearer picture on how to conduct their simulations. Currently, the PI has requested to put the project on hold.

Project 128	**EHPC-DEV-2025D05-006**	CINECA
Machines: Leonardo		Collaborators: IT4I@VSB
Project goals	The primary goal of the project was to run a GPU-accelerated Fortran code (Traf), using MPI/OpenMP, OpenACC, and CUDA Fortran, on the Leonardo supercomputer.	
Work done	The PI provided two different compilations (one with GNU-OpenMPI and the other with NVHPCX), which were evaluated by profiling a benchmark run using NVIDIA Nsight Systems. The profiling results revealed no significant difference in performance between the two MPI implementations, likely due to the absence of collective communications in the code. The Thomas algorithm kernel, responsible for data smoothing, was identified as the most time-consuming function, occupying over 55% of the total GPU time. Based on these findings, we agreed to proceed with strong scaling tests and focus on optimizing the smoothing routine. The PI provided the source file for the smoothing kernel along with the object files for the rest of the code. In the strong scaling tests for small and medium-sized systems, we identified the optimal number of nodes for efficient scaling. For the small system, efficient scaling was achieved up to 16 nodes, with performance above 70% of the perfect linear scaling. For the medium-sized system, this limit increased to 32 nodes. We recommended using OpenMP thread bindings to avoid thread migration, particularly at higher node counts, and found that communication time was dominated by point-to-point communication. Additionally, we observed that broadcast communications were limited to the CPU and had minimal impact on performance, even with larger node counts. Optimizing communication, especially point-to-point and broadcast operations, was identified as key to improving scalability. The profiling also pointed out that some communications were not fully utilizing the GPU, indicating possible inefficiencies in the CUDA-aware communication set-up.	
Improvements provided	We provided the PI with the optimal number of nodes for their simulations (16 to 32), along with the best configurations possible for their scripts.	
Conclusions	We demonstrated and learned that the MPI implementations (OpenMPI and NVHPCX) did not affect the performance of the code. We found that the limiting step in the program was the Thomas algorithm kernel, occupying 55% of the total GPU time. This kind of porting using OpenACC kernels scheduled via OpenMP presents challenges to the usual NVIDIA profilers, which made the analysis challenging.	

Project 127	**EHPC-REG-2024R02-151**	LUXPROVIDE SA
Machines: MeluXina		Collaborators: None
Project goals	The goal of the project was to investigate and address scalability issues encountered by the PI team in the Finite Elements Method (FEM) software Schism. Specifically, the PI team reported that Schism could scale effectively up	

	to 1024 cores, but performance degraded significantly beyond that. The objective was to identify the root cause of this issue and propose solutions to improve scalability on large core counts.
Work done	The first step involved obtaining the source code and reproducing the reported performance problem. We identified that the scaling issue was particularly evident during the force preparation steps, which struggled to scale as the number of tasks increased. We tested four different workloads for 4, 6, 8 and 10 nodes each (with 128 cores per node). After discussions with the PI team, it became apparent that the issue was related to parallel I/O, with Schism generating many small file read/writes and created a bottleneck on the Lustre file system. To confirm whether the file system was the root cause, we ran all 12 tests again by executing the code entirely in memory, bypassing Lustre. However, the issue persisted, indicating that the problem was not directly related to Lustre. We further investigated the MPI collective operations in the force_prep procedure of the Fortran source code. It appeared that this section of the code relied heavily on collective operations, which may have been contributing to the scalability limitations. We suggested to the PI team that they utilize optimized collectives available in HPCX, which could be loaded dynamically without needing to rebuild the code. This provided much better results for all 12 tests, and an improved scalability in general.
Improvements provided	Our work resulted in a better scalability for the Schism code, enabling users to use a larger number of CPUs to greatly reduce the time to solution.
Conclusions	The investigation revealed that the scalability issue in Schism was primarily linked to inefficient collective operations, rather than the Lustre file system itself. By utilizing optimized collectives from HPCX, we achieved better scaling in the force preparation steps, though further tuning may be needed to fully optimize the performance for very large core counts.

Project 126	**EHPC-BEN-2025B05-003**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	Get a working setup for Machine Learned-based Particle Flow (MLPF, https://github.com/selvaggi/mlpf/tree/main).	
Work done	The work involved setting up the Conda environment for the users, which included non-standard Python packages that were not available in pip repositories (pytorch_cmspepr and geometric-algebra-transformer). We then provided a way to set up WandB offline to monitor the training of the neural network. Getting this set-up to work required several alterations due to the complexity of the training workflow and included re-adapting the workflow to avoid outbound calls from the compute node into the internet.	
Improvements provided	We provided the entire set-up for researchers to run with, thus enabling their research and saving them weeks of trial and error.	
Conclusions	We managed to create a set-up that allows running WandB offline. This served as a general guideline for the projects that arrived after this project. It has been made available in the EPICURE software repo: https://opencode.it4i.eu/epicure/epicure-sw/-/tree/main/support-tools/wandb-offline	

Project 125	**EHPC-DEV-2025D02-093**	CSC
Machines: LUMI, Leonardo, MareNostrum 5		Collaborators: None
Project goals	Improve performance for gPLUTO, a software package coming from the SPACE CoE (Centre of Excellence).	
Work done	We could not work directly on the code itself, but we provided researchers with support for profiling and configuring their run scripts. Support was limited to LUMI, and after several exchanges with the PI, they managed to get their program running in a multi-node, multi-GPU scheme.	
Improvements provided	Enabled running of gPLUTO under a multi-GPU scheme in LUMI.	
Conclusions	Support was limited to a few exchanges with the PI. Researchers were able to run gPLUTO with multi-GPU and multi-node support.	

Project 123	**EHPC-DEV-2025D01-005**	SofiaTech
Machines: Discoverer		Collaborators: None
Project goals	The primary goal of the project was to port, modernize, and optimize a Fortran-based quantum mechanical simulation code provided by the PI. This included ensuring platform independence, enhancing scalability, and integrating MPI and OpenMP hybrid parallelism for improved performance on EuroHPC systems. The code needed to be modified for efficient execution on different CPUs, with compatibility for various BLAS implementations and a focus on achieving optimal performance across diverse hardware architectures.	
Work done	The first task involved porting the code by formatting and organizing it for better readability, then selecting an appropriate Fortran compiler and linker. We opted for the LLVM 20 toolchain, utilizing its Flang Fortran compiler for improved compliance with Fortran 2018 standards. A build script system was created, as the original code lacked one, and necessary compiler flags were defined to ensure portability. The code was aligned with the goal of platform independence, ensuring it could compile and run on various EuroHPC systems without modification. Additionally, we modernized the code to follow standard Fortran modular design, enhanced OpenMP parallelization, and ensured compatibility with multi-threaded BLAS libraries for better computational efficiency. This included adjusting OpenMP threading, memory access optimization, and thread safety improvements. For scalability, the code, initially designed for 32 threads with OpenMP, was extended to support hybrid MPI+OpenMP parallelism. This was crucial since the original code could not scale beyond a single compute node. With the integration of MPI, we enabled the code to run efficiently across multiple nodes, drastically improving simulation runtimes.	
Improvements provided	We provided a modernized version of the code that was able to increase scalability by incorporating MPI support. Not only that, but the entire process resulted in a code that can be supported by a wider range of compilers and architectures, and that is easier to maintain. This enables the users not only to produce faster results (getting great speed-ups from MPI usage in larger amounts of cores) but also might enable new science for them.	
Conclusions	The porting process itself significantly improved code portability by making it more architecture-agnostic, allowing it to run on various hardware configurations	

	such as x86 (for Intel and AMD EPYC processors) and ARM. The use of LLVM's Flang compiler helped achieve compliance with modern Fortran standards, making the code more maintainable and compatible with future compilers. We enhanced the OpenMP parallelization for better load balancing, reduced thread synchronization overhead, and ensured that numerical stability was maintained. The optimization of memory access patterns and thread safety reduced race conditions and contention, further improving performance. Through the integration of MPI, the code was able to scale across multiple nodes, which was not possible with the original OpenMP-only implementation. The careful choice of multi-threaded BLAS libraries optimized performance, though limitations in OpenBLAS for certain routines (like zheev) were noted, prompting further evaluation of alternatives such as ScaLAPACK and GPU-accelerated libraries like CULA Tools.

Project 122	**EHPC-DEV-2025D04-139**	LUXPROVIDE SA
Machines: MeluXina		Collaborators: None
Project goals	To identify performance bottlenecks in the baseline distributed RoBERTa training pipeline using detailed profiling tools (PyTorch Profiler and TensorBoard Trace Viewer). The goal is to analyse compute-communication patterns, GPU utilization, and data flow to guide targeted optimizations.	
Work done	The first training script we used showed signs of several communication bottlenecks, as it contained a serialized all_reduce operation after the backward pass. We identified this routine in trace analysis, which started the optimization process. We adjusted DDP bucket_cap_mb to trigger earlier syncs (for the all_reduce operations), we enabled Gradient Accumulation, and we added FP16 compression hooks to reduce bandwidth, and tuned NCCL and other environment variables to improve pipeline behaviour. Summarizing, the main optimization steps were: • Identified the presence of delayed all_reduce operations in trace • Adjusted DDP bucket_cap_mb to trigger earlier syncs • Added FP16 compression hooks to reduce bandwidth • Tuned NCCL and environment variables to improve pipeline behaviour • Verified with trace of optimized code that all_reduce operations became interleaved with backward	
Improvements provided	The optimizations reduced peak memory usage by a 25% (from 32GB to 24GB). In addition, the optimizations enabled better scaling and a speed-up of 4x times when compared with the initial set-up.	
Conclusions	By tuning several DDP options, we managed to get a better scalability and 4x speed-up with respect to the original configuration. Compressing data to FP16 greatly reduced bandwidth stress, making it the most impactful among all the changes.	

Project 121	**EHPC-DEV-2025D04-116**	IT4I@VSB
Machines: Karolina		Collaborators: None
Project goals	The goal of this project was to enable the PI to access and run their Python	

	AI/ML workloads on Karolina. An additional objective was to assist in designing an RTSP video streaming solution to an accelerated compute node.
Work done	Our first session with the PI covered HPC best practices, access to the machine, and other essentials to handle their jobs in Karolina. Afterwards, we prepared the environment required for their AI/ML workloads. We also demonstrated how to get models via the Hugging Face CLI for future set-ups. Finally, we provided guidance on how to design the injection of an RTSP (Real Time Streaming Protocol) stream from a webcam into the compute nodes in real time, but no end-to-end test was performed.
Improvements provided	The key improvement was enabling the PI to do their research on the machine, by getting their environment set-up and providing guidance on how to use the machine.
Conclusions	The engagement successfully achieved the primary goal of enabling the PI to access and use Karolina independently, providing the foundation for associated future work.

Project 118	**EHPC-DEV-2025D05-093**	UAntwerpen
Machines: LUMI, Leonardo		Collaborators: CINECA, CYFRONET, IT4I@VSB
Project goals	The primary goal of this engagement was to address and resolve multiple performance and functionality issues encountered by the Principal Investigator (PI) when using MPI and GPU-aware communication on both the LEONARDO and LUMI HPC systems. Specific tasks included improving performance with non-contiguous data types, troubleshooting GPU-aware MPI on LUMI, enhancing profiling tools for MPI and GPU interactions, and investigating issues with intra-node asynchronous communications. Additionally, the project aimed to support the PI in optimizing these workflows for scalable performance and correct data transfers.	
Work done	On LEONARDO, the PI reported performance degradation with non-contiguous data types. The NVIDIA contact at Leonardo compiled a version of MPICH, but the new library caused a segmentation fault, requiring further adjustments. We decided to await the PI's tests using contiguous data and the updated MPICH version before proceeding further. On LUMI, the PI initially requested a solution to non-contiguous data types as well, but this was postponed to focus on other issues. The team worked on solving GPU-aware MPI errors, which were traced to incorrect data transfers due to Slurm configuration. A fix using the ROCR_VISIBLE_DEVICES environment variable was proposed and tested, leading to the resolution of data transfer issues, though multi-node configurations continued to cause errors. Further adjustments, such as adding synchronization steps (hipDeviceSynchronize), were implemented to resolve issues with page faults on multi-node runs. Profiling tools were also addressed on LUMI, with the PI requesting a better MPI and GPU tracing solution. Although the omnitrace tool worked, it had a less user-friendly workflow compared to NVIDIA NSIGHT. We provided guidance on using omnitrace with Slurm and troubleshooting results via Perfetto. The PI also requested multi-node tracing support, which required further investigation into AMD's default settings. Other ongoing tasks included investigating intra-node	

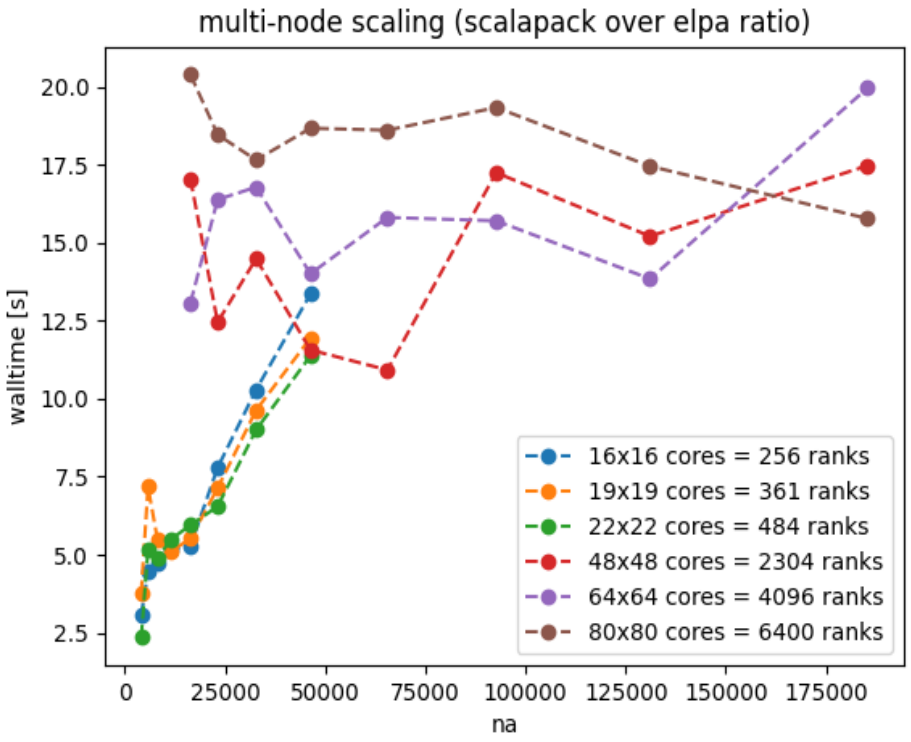
	asynchronous communications, with the PI observing delays in MPI Isend/Irecv operations. Solutions were proposed to modify buffer sizes and communication configurations, and tests were conducted, although final resolutions were deferred to a related EPSILOD project.
Improvements provided	The project made significant progress in troubleshooting and optimizing MPI and GPU workflows. On LUMI, the troubleshooting of GPU-aware MPI communication significantly improved data transfer, with a configuration fix for Slurm improving performance and the addition of synchronization steps preventing page faults.
Conclusions	The problem with non-contiguous data types on LEONARDO and LUMI was identified, but full resolution depends on the PI's further tests with contiguous data and the updated MPICH version. On LUMI, the fixes for GPU-aware MPI communication proved effective in improving performance but highlighted ongoing complexities with multi-node configurations and asynchronous data transfers.

Project 116	**EHPC-BEN-2025B03-064**	IZUM
Machines: Vega		Collaborators: JSI
Project goals	The goal of this project was to address the issues the users faced when running LLaMA AI models via GPT-J; mainly, the usage of multi-GPU set-ups.	
Work done	We started working on a set-up for running GPT-J in a multi-GPU scheme. By backtracing the errors, we found out several indexing errors within the model itself, which seemed like an internal PyTorch issue. We contacted the PI to establish a potential solution, but we did not receive any answer.	
Improvements provided	We could not assess whether our investigations yielded any improvements for the users, since the PI did not reply.	
Conclusions	Fixing issues with inference models is not trivial due to the many processes involved. In addition, it has become apparent that not all Hugging Face models can work straight out of the box on all systems.	

Project 115	**EHPC-DEV-2025D03-071**	KTH
Machines: LUMI		Collaborators: None
Project goals	The primary goal of this engagement was to support the profiling and benchmarking of a miniapp (a miniapp is a small, self-contained program that captures the core computational patterns of a much bigger code) from EMTO code (https://emto.gitlab.io/) on the LUMI HPC system.	
Work done	The first task involved assisting the user with building and compiling the miniapp on LUMI, specifically using Cray Fortran. This included providing advice on troubleshooting issues related to hipFORT interfaces, helping the user build an updated version of hipFORT with the latest ROCm release available on LUMI. Another key effort was dedicated to building an updated version of the MAGMA library (2.8.0) for LUMI. However, this attempt was unsuccessful, as testing revealed that the ROCm version of MAGMA performed better. Further investigation identified the root cause of build errors, which turned out to be related to the use of the newgrp command to switch active user groups, inadvertently clearing the necessary environment variables for Cray Fortran's	

	error message system. We tested the latest version of AMD's Omniperf profiling tools (as part of ROCm 6.2.2) and provided the user with detailed instructions for their use. The tools required an extra environment variable, ROCPROF=rocprofv2, to enable hardware counters to function correctly. Basic profiling and analysis were conducted on the miniapp's major kernels, including a compute intensity and roofline analysis to assess performance bottlenecks.
Improvements provided	The main improvements provided included updates to the miniapp's build process, specifically with hipFORT and ROCm libraries, which facilitated smoother compilation and execution on LUMI. This paves the way to get the full code installed in an optimal way in LUMI
Conclusions	We successfully addressed key build and performance challenges related to the miniapp on LUMI, while the attempt to build an updated version of MAGMA was abandoned due to performance concerns. The transition to using the ROCm-native MAGMA library proved beneficial. Profiling with Omniperf offered important insights into the miniapp's performance, helping the user optimize specific kernels for better efficiency.

Project 114	**EHPC-DEV-2025D04-097**	UAntwerpen
Machines: MareNostrum 5		Collaborators: None
Project goals	The major goal was to estimate the speedup to be gained from replacing ScaLAPACK for solving eigenvalue problems in MOCCa by ELPA. Currently solving eigenvalue problems takes between 40 and 70% of the compute time for MOCCa.	
Work done	We conducted several tests to explore the weak and strong scaling for eigenvalue problem solvers. We installed and tested ELPA, relying on the python wrappers PyScaLAPACK and PyELPA. We tested different configurations. Up to 50000 elements (7 tests for each set of ranks), we tested 256, 361, and 484 MPI ranks. Then from 20000 to 175000 elements, we tested 2304, 4096 and 6400 ranks (8 tests each). We found out that ELPA significantly outperforms ScaLAPACK in all instances, from 2.5x in the smallest problems to up-to 20x in larger problems and ranks.	
Improvements provided	The outcomes of this project showed that, for the cases studied, MOCCa should always prefer ELPA over ScaLAPACK for its linear algebra routines. These results were shared with the MOCCa team, leading to a follow-up project to improve MOCCa based on these observations.	

	 *6 - Figure showing the speed-up of using ELPA instead of ScaLAPACK.*
Conclusions	We found that ELPA significantly outperforms ScaLAPACK. In addition, all information gathered through the project was shared with the MOCCa development team, leading to work on a partial rewrite of MOCCa that introduces a Python front end to facilitate integration with external libraries, particularly HPC libraries used for computationally intensive tasks.

Project 113	**EHPC-DEV-2025D02-094**	CYFRONET
Machines: LUMI		Collaborators: KTH
Project goals	The project aimed to optimize the performance of the proprietary astrophysics code THATMPI (a Fortran-based PIC code using MPI+OpenMP and HDF5) on the LUMI supercomputer. Initially framed as a "code porting" effort, the focus shifted to addressing performance bottlenecks and improving efficiency.	
Work done	The team began by evaluating THATMPI's performance across three machines: LUMI (AMD Zen3), Ares (Intel Cascade Lake), and Helios (AMD Zen4). Initial tests revealed inconsistencies in the provided performance data, prompting the creation of standardized test configurations and batch scripts for LUMI, Ares, and Helios. The Makefile was corrected and simplified, ensuring compatibility with LUMI and enabling compilation with both ifx and ifort. Intel compilers and MPI were verified and configured correctly on LUMI. Performance tests were conducted using various compiler optimization flags. It was noticed that the PI was using -xHost on AMD which is known to result in degraded performance. The tests showed a consistent improvement using -O3. The team also implemented performance metrics, including simulation rates (Mpart/s) and detailed timing breakdowns for critical functions like particle pushing and deposition. These metrics were integrated into the simulation	

	output, enabling more accurate benchmarking. A new benchmark was developed to compare Intel's legacy (ifort) and new LLVM-based (ifx) compilers, as well as the impact of AVX instructions. Tests confirmed that -xHost was detrimental for AMD CPUs, while AVX optimizations provided a ~1.25x speed-up. The optimal flags identified were -O3 -arch AVX for ifort and -O3 -xHost for ifx. A proof-of-concept port to Cray and gfortran compilers was also initiated, though runtime memory issues remain unresolved. Given the substantial performance gains achieved with Intel compilers, further porting efforts were deprioritized.
Improvements provided	The project delivered significant performance improvements, achieving a ~5.1x speed-up on LUMI by switching from -O3 -xHost to -O3 -arch AVX for ifort. The optimal compilation flags were identified for both ifort and ifx, and a robust benchmarking framework was established. This framework now includes reference performance data for multiple machines (Ares, LUMI, Helios) with different CPU architectures, simplifying future optimization and development efforts.
Conclusions	The project successfully addressed most of the principal investigator's requests, with a focus on performance optimization rather than extensive porting. The root cause of poor performance on LUMI was identified as incorrect compiler flags, and switching to -O3 -arch AVX yielded a 5.1x improvement.

Project 112	**EHPC-BEN-2025B04-049**	CINECA
Machines: Leonardo		Collaborators: None
Project goals	Provide the user with general support for PyTorch on Leonardo.	
Work done	The work under this project involved providing guidance on how to use PyTorch for AI training on Leonardo. General advice on Slurm and file system usage was provided, with special attention to multi-node processing.	
Improvements provided	We provided general guidance to the users to get their projects running speedily, essentially reducing the set-up time for their simulations by days.	
Conclusions	We successfully provided the users with general guidance on AI training in Leonardo using Python.	

Project 111	**EHPC-BEN-2025B02-048**	CYFRONET
Machines: LUMI		Collaborators: None
Project goals	The goal of this project was to assist the PI in optimizing the performance of Quantum Espresso (QE) and Yambo simulations, specifically by analysing and improving the resource usage for large-scale calculations. The project aimed to develop a structured set of tests to benchmark resource usage and create tools for memory measurement.	
Work done	The project began by gathering and analysing the input parameters for QE and Yambo, then running a sample simulation workflow provided by the PI. This helped in understanding the performance bottlenecks and resource usage. A simple tool for measuring memory usage was created, and discussions were held with the PI to define the approach for benchmarking the resource usage. The PI developed a set of tests focusing on varying the number of atoms and	

	other parameters that affect computational complexity. A streamlined nine-step workflow, including batch scripts for both the PI's base cluster at Cyfronet and the LUMI system, was created to facilitate testing with memory measurement enabled. Additionally, the existing tool at Cyfronet, hpc-jobs-history, was extended to report memory and CPU usage based on data from sacct. The PI was assisted in running the full set of tests, and the results were collected and analysed in a spreadsheet. Tests were performed on different configurations selected by the PI, including changes to parameters like xs, ex, k, and eh. Scalability and parallelization tests for Yambo were conducted, particularly for the polarisation step (s5-eps), with different CPU role settings. Similarly, scalability tests for QE's nscf step were carried out. Resource configurations were fine-tuned to run and monitor large Yambo calculations for the PI's test case soc-96-big on LUMI, which involved using 128 nodes over two days. The results were documented in a detailed support report, which included resource usage charts and conclusions.
Improvements provided	Key improvements included creating a more efficient testing and benchmarking workflow, enabling the PI to measure and analyse memory usage alongside CPU performance. This made it easier to identify resource bottlenecks and optimize the simulation parameters. By extending the Cyfronet hpc-jobs-history tool, more detailed reports on memory and CPU usage were provided, helping the PI better understand how resources were being consumed across different simulations.
Conclusions	We found that for Yambo, more MPI tasks resulted in much more memory being used, but it also reduced the time to solution. This could be partially mitigated by using OpenMP, but the speed-up was lower than using pure MPI.

Project 110	**EHPC-AI-2024A02-044**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	Providing a set-up to run multi-node AI training on NVIDIA H100 GPUs, using PyTorch and Hugging Face Accelerate.	
Work done	The work focused on setting up and validating the software environment required for multi-node AI training on NVIDIA H100 GPUs, including the installation and configuration of PyTorch, Hugging Face Accelerate, and the Hugging Face Datasets library. The activities comprised collecting the dataset and source code provided by the user and preparing the necessary job scripts to launch distributed training. Detailed performance optimization and AI-specific enhancements were outside the scope of WP2 and were addressed in WP3, which is why the overall effort in WP2 was minimal for this project.	
Improvements provided	We provided the set-up for the users to run their training, which paved the way for further optimizations done under WP3.	
Conclusions	We successfully provided the users with the required modules and scripts to run their AI training.	

Project 109	**EHPC-DEV-2025D04-113**	CINECA
Machines: Leonardo		Collaborators: None
Project goals	The user faced very low GPU usage with SPUMA when running multi-node training. The aim is to get insights on how to improve the performance of SPUMA, which will be done as part of WP3.	
Work done	We modified the initial set-up provided by the user, to run the benchmark (80M cells) in a multi-node configuration on 8 GPUs of Leonardo. AmgX has been used as linear algebra package to solve the pressure equation, but we also tested the native GAMG linear solver.	
Improvements provided	We provided a tested GPU-enabled installation for SPUMA, while also providing guidance on how to run the simulations.	
Conclusions	We successfully ran the benchmark for 6k iterations on 8 GPUs in about 11k seconds (3h). The solver is about 10 times faster than 288 cores (Intel(R) Xeon(R) Platinum 8160) hosted in the ENEA Cresco6 infrastructure (located in Portici).	

Project 107	**EHPC-DEV-2025D04-021**	CINES
Machines: LUMI		Collaborators: None
Project goals	The primary objective of this task was to address performance discrepancies between AMD and NVIDIA hardware when running the PI's code on different platforms. The goal was to optimize the code to match the performance seen on the PI's workstation and reduce the significant performance gap observed between CUDA on NVIDIA GPUs and HIP on AMD hardware.	
Work done	In Task 2.1, we began by testing the PI's code across various hardware platforms to assess the performance differences between CUDA and HIP. After reproducing the reported performance results, we made several adjustments to the compilation parameters without altering the code itself. These adjustments were aimed at better aligning with the specific hardware and software configurations of the various platforms. This optimization resulted in a significant reduction in runtime on LUMI MI250x, from 38 minutes to 50 seconds, and further down to approximately 30 seconds, nearing the performance observed on NVIDIA hardware. In Task 2.2, we focused on analysing the performance issues tied to AMD hardware. We used profiling tools like rocprov3 to examine kernel dispatches and inspect the HIP runtime protocols. A key discovery was that the use of hipMallocManaged on AMD hardware resulted in suboptimal performance. While it performs somewhat better in CUDA, on HIP it triggered frequent and unnecessary data transfers between the CPU and GPU, contributing to a large part of the execution time. To address this, we proposed the development of a GPU-only version of the application to eliminate the need for hipMallocManaged and avoid unnecessary host-device memory transfers. The GPU-porting process done in WP3 was continuously monitored with profiling tools to visualize the impact of changes and to track the remaining optimization opportunities, which is part of WP2 tasks. The profiling traces, both from the initial and post-porting stages, helped to visualize the improvements, showing that memory transfers were significantly reduced, and that kernel execution was running more concurrently with the introduction of GPU threads	

	and streams.
Improvements provided	The optimization works significantly improved the performance of the code on LUMI MI250x, reducing runtime from 38 minutes to around 30 seconds (~76x speedup), closely matching the performance of NVIDIA hardware. This was achieved through careful adjustments to compilation parameters and the configuration of the Unified Memory environment.
Conclusions	The project successfully bridged the performance gap between NVIDIA and AMD hardware, improving the execution time on LUMI MI250x and achieving performance that was more aligned with the results seen on NVIDIA hardware. The root cause of the performance bottleneck, hipMallocManaged, was identified, and a more efficient memory management approach was implemented. By porting the code to a GPU-only version, we eliminated unnecessary host-device memory transfers, resulting in significant performance improvements.

Project 106	**EHPC-DEV-2025D04-077**	LUXPROVIDE SA
Machines: MeluXina		Collaborators: None
Project goals	Provide the researchers with guidance and scripts to perform their AI model training on MeluXina.	
Work done	Researchers provided the team with two models to train, eubert_training and eurovoc_training. We made a launcher script for both models, which allowed an efficient usage of GPUs.	
Improvements provided	We enabled the users to run their training on MeluXina, saving weeks of set-up.	
Conclusions	We provided launcher scripts for the AI trainings that we can now also share with other users for similar cases.	

Project 105	**EHPC-DEV-2024D12-048**	CINECA
Machines: Leonardo		Collaborators: None
Project goals	Help the PI run their AI jobs on Leonardo.	
Work done	We provided details on how to run AI trainings on Leonardo, while also supporting the PI on setting up IDEs such as Jupyter.	
Improvements provided	We provided the PI with the information required, but soon after they required to put the project on hold, so it's difficult to gauge the full impact of the support provided.	
Conclusions	We provided support on how to set up AI jobs and IDEs in Leonardo.	

Project 104	**EHPC-DEV-2025D01-065**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	The project aimed to support the porting of the HPCW benchmark suite to both the ACC and GPP partitions of MareNostrum 5. The primary objective was to ensure that the Earth science miniapps within the benchmark suite ran efficiently on both systems.	

Work done	The project successfully built the HPCW benchmark suite, making necessary adjustments to the build process for both the Intel and NVHPCX toolchains. Notably, the NEMO component was temporarily excluded due to an ongoing major overhaul. The team adapted the dependency management process, removing references to NEMO and its dependencies, such as XIOS, as per project instructions. Benchmark deployment was carried out on both the General Purpose Partition (GPP) and the Accelerated Partition (ACC) of MareNostrum 5. On the GPP, successful tests and builds were achieved for a range of miniapps, including multiple ICON configurations, ectrans, CLOUDSC, ecRad, and nicamdc. On the ACC partition, the team also completed successful tests and builds for ICON, ectrans, and CLOUDSC, demonstrating compatibility and performance across different hardware configurations.
Improvements provided	The project delivered a robust and adaptable build process for the HPCW benchmark suite, tailored to the specific requirements of MareNostrum 5's partitions.
Conclusions	The project met its core objectives by porting and validating the HPCW benchmark suite on MareNostrum 5's partitions, despite the temporary exclusion of NEMO. The work not only confirmed the suite's compatibility with MN5 architectures but also generated actionable performance data.

Project 103	**EHPC-REG-2024R02-178**	CINECA
Machines: Leonardo		Collaborators: UAntwerpen, INESC TEC
Project goals	The objective was to provide an installation of the HeFFT library that the PI wanted to add to their code. They also wanted a memory bandwidth benchmark to compare float4 against float3 data types.	
Work done	We proceeded with the installation of HeFFT on Leonardo and Deucalion, while providing small memory benchmarks for the code.	
Improvements provided	We enabled the PI's code by providing the library installation.	
Conclusions	HeFFT needed to be compiled on compute nodes due to specific library requirements. While A100 GPUs were better suited for the float4 than the float3 datatypes, the difference in speed was marginal (around 1.2%).	

Project 102	**EHPC-REG-2024R02-131**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	The project aimed to establish a fully functional installation of CyberShake on both the General Purpose Partition (GPP) and the Accelerated Partition (ACC) of MareNostrum 5. The primary objectives were to port the existing workflow from MareNostrum 4, address compatibility issues, and to update the system to support the latest version of CyberShake (5.5.2).	
Work done	The team began by installing CyberShake on the GPP following provided documentation. Initial testing of the workflow, originally used on MareNostrum 4 revealed several checkpoints where manual intervention was required. During the execution of DirectSynth, issues arose with the generation of GRM and SRF	

	files. After investigating and modifying the relevant code—including un-commenting and adjusting the SRF generation routine—the team successfully produced both files. In parallel, the workflow was updated to support CyberShake version 5.5.2, while maintaining compatibility with version 5.4.2. Each step was thoroughly documented. The team then focused on the ACC installation, where most files required no changes. However, updates were necessary in the AWP-GPU-SGT directory to remove deprecated texture functions for CUDA compatibility. Job submission scripts were revised, replacing aprun with srun and adjusting resource allocation. The set-up was validated with a successful test model, and all changes were committed to a GitHub repository for version control.
Improvements provided	The project delivered a robust and updated installation of CyberShake on both GPP and ACC partitions of MareNostrum 5, thus enabling user access to them.
Conclusions	Key improvements included resolving compatibility issues with modern toolchains, automating previous (manual) workflow steps, and updating job submission scripts for the new system. The team also enhanced version control and documentation, making the process more reproducible and accessible.

Project 99	**EHPC-REG-2024R02-140**	CSC
Machines: LUMI		Collaborators: None
Project goals	The goal here was to provide optimal bindings and submission scripts for an already containerized piece of software.	
Work done	Most of the expensive functions in the software itself were already using jit compilation, furthermore the code base makes wide usage of NumPy/PyTorch. Therefore, there is not much room for optimization. In addition, the problem is "data parallel", i.e. mostly embarrassingly parallel, so synchronization between ranks is not a problem. All in all, the modifications to the code that were made mostly didn't target "raw performance", but better usability and scalability on a supercomputer, as the code was not designed for it. We compiled a few key packages from source (such as NumPy and PyTorch), but gains were marginal.	
Improvements provided	We extended the original code for usage under HPC, mainly setting up the correct bindings and fine-tuning multi-node usage.	
Conclusions	Even when the code is mostly done by the original developers, there was plenty of room for improvement in terms of HPC usage. We want to highlight the help from CINECA with their profiling tools which worked out-of-the-box on LUMI. This wouldn’t have been possible without the already established collaborations via EPICURE.	

Project 98	**EHPC-DEV-2024D11-038**	IT4I@VSB
Machines: Leonardo		Collaborators: CINECA
Project goals	The project aimed to streamline and optimize molecular dynamics (MD) simulation workflows using AMBER on the Leonardo cluster. The primary objectives were to automate job management, ensure efficient handling of simulation outputs and data, diagnose and resolve technical errors, and address	

	cluster and software-specific issues.
Work done	The team developed and provided Slurm job dependency scripting templates to automate chains of dependent MD simulations, explaining the logic for job continuation and custom loops to accommodate varying simulation lengths. Guidance was offered on AMBER's restart strategy and the correct chaining of runs using md_*.rst7 files, with resource allocation verified for compatibility with the Leonardo cluster architecture. For simulation output and data handling, the team clarified trajectory concatenation procedures and supplied a cpptraj script for merging .nc files. Users received advice on storage strategies, file system best practices, and quota management tools. In error diagnosis and troubleshooting, the team identified and corrected issues related to missing or misused restraint files, supported the set-up of restrained MD simulations, and ensured proper module loading and environment configuration. Tailored scripts were provided for restarting unfinished jobs, and follow-up meetings were scheduled to address simulation stalls. Cluster and software issues were investigated, including simulation freezing under GPU usage, Slurm script validation, and post-outage trajectory minimization.
Improvements provided	The project delivered ready-to-use scripting templates and job dependency management tools, significantly streamlining MD workflows and improving reproducibility. Resource allocation was optimized for the Leonardo cluster, preventing scheduling inefficiencies. Early diagnosis of common user errors, such as missing restraint files and incorrect input flags, reduced simulation failures and accelerated user learning.
Conclusions	The project successfully automated and optimized MD simulation workflows on the Leonardo cluster. All the scripts are available in the next link: https://opencode.it4i.eu/epicure/ehpc-dev-2024d11-038

Project 97	**EHPC-BEN-2024B11-044**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	The project aimed to benchmark and optimize the performance of GPU-accelerated solvers in GronOR. The goal was to identify why cuSOLVER under-performed compared to CPU-based solvers (LAPACK and EISPACK) and to implement improvements that would bring its execution time in line with or better than the alternatives.	
Work done	The team began by reviewing the user's job scripts and modifying them to enable NVIDIA's Multi-Process Service (MPS), allowing up to two MPI processes per GPU. This change alone reduced cuSOLVER's execution time by 37.5%, from 107 seconds to 66.9 seconds. However, even with this improvement, total cuSOLVER time (166.31s) remained significantly higher than LAPACK (37.2s) and EISPACK (28.07s). Further investigation using NSIGHT profiling tools revealed that cuSOLVER was not fully utilizing GPU resources, suggesting a possible limitation with small matrix inputs. Additionally, the team discovered that the CMakeLists file contained hard-coded cuSOLVER link paths for another cluster. After adjusting	

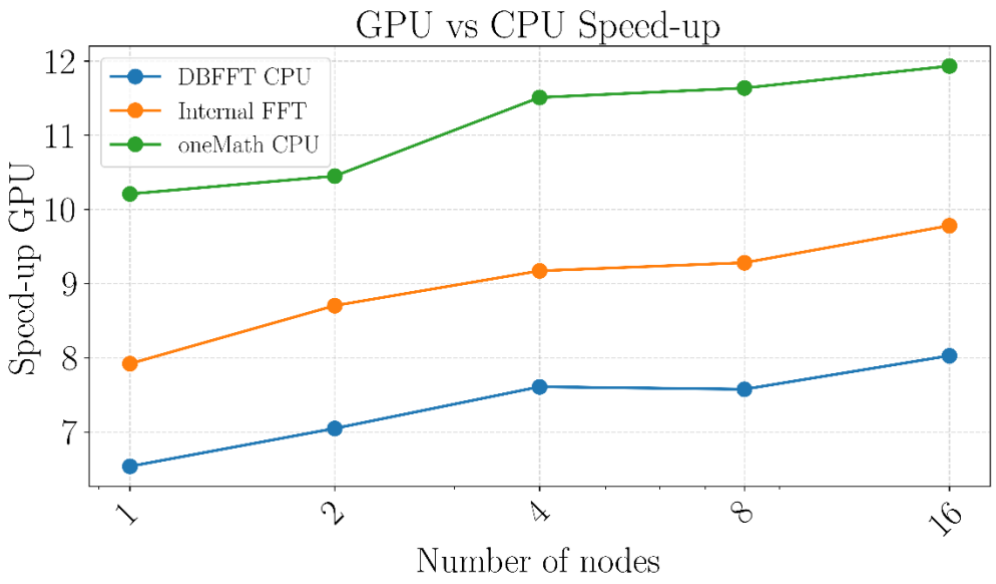
	these paths to match MareNostrum 5's environment and recompiling, cuSOLVER's performance improved further, achieving 101.99 seconds with four ranks per GPU—though still not matching the speed of the CPU solver.
Improvements provided	The project delivered several key improvements: enabling MPS reduced cuSOLVER's runtime by over a third and correcting the hard-coded link paths in CMakeLists led to additional performance gains. Profiling with NSIGHT identified under-utilization of GPU resources, pointing to potential optimizations for small matrix operations.
Conclusions	While cuSOLVER showed improvement after enabling MPS and correcting build paths, its performance remained below that of CPU-based solvers for the tested matrix sizes. The results suggest that for small matrices, alternative solvers or algorithms may be more efficient, and that further tuning—such as exploring Jacobi methods or larger matrix workloads—could yield better outcomes.

Project 96	**EHPC-DEV-2025D12-017**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	To provide guidance and advise on improving the profiling and benchmarking already done for Horses3D.	
Work done	We made use of JUBE to adapt and improve the benchmarking scripts provided by the users. We provided back scripts that tested different mesh sizes and different p-orders.	
Improvements provided	Improved the benchmark scripts provided by the users by automating them in JUBE, making them easier to extend.	
Conclusions	We provided extended versions of the benchmarks to the users, allowing them to have a better understanding of their solver.	

Project 95	**AI-BOOST**	CYFRONET
Machines: LUMI		Collaborators: None
Project goals	The project aimed to optimize the performance and scalability of OLMo training using PyTorch on LUMI's AMD MI250x GPU infrastructure. The primary objectives were to benchmark communication performance using the IBM PyTorch communication benchmarks, resolve scalability issues up to 64 nodes (512 logical GPUs), and identify and fix bottlenecks related to library paths, network communication, and container configurations.	
Work done	Initial tests focused on communication performance using the IBM PyTorch benchmarks and the aws-ofi-rccl plugin, revealing issues with library paths for libcxi.so and libjson-c.so. Enabling libfabric CXI and aws-ofi-rccl significantly improved token throughput, from ~300 tokens/s per device without CXI to over 1200 tokens/s at smaller scales, though performance degraded at 64 nodes (~500 tokens/s). The team prepared scripts and configurations for systematic testing, exploring various training settings (compilation modes, flash-attention, batch sizes) and NCCL network flags. With a new container (torch2.6+rocm6.2.4), throughput improved further, reaching ~1500 tokens/s at smaller scales and ~1400 tokens/s at 32 nodes (total ~330k tokens/s), with 64-node runs achieving ~1100 tokens/s (~570k total throughput). Some jobs, especially at larger scales, experienced	

	hangs before training, prompting tests with CPU binding and consultations with AMD/HPE support. The final phase involved testing a new LUMI container (lumi-pytorch-rocm-6.2.4-python-3.12-pytorch-v2.6.0.sif), which enabled stable 64-node training at ~700k tokens/s total (~1370 tokens/s per GPU) using "default" compilation and optimized batch sizes. The team also investigated sharding strategies for scaling beyond 64 nodes and resolved performance drops caused by container symlink changes.
Improvements provided	The project delivered a robust environment for multi-node OLMo training, achieving scalable performance up to 64 nodes, and providing at least a 4x speed up when compared to the initial set-up for the trainings.
Conclusions	Upgrading core libraries like PyTorch yielded significant "free" performance gains, while careful management of compilation caches and network settings proved essential for scalability.

Project 94	**EHPC-DEV-2024D12-094**	CINES
Machines: LUMI		Collaborators: None
Project goals	The project aimed to enable the BigDFT code to run on AMD GPU platforms, specifically on LUMI's GPU partition, by adapting its SYCL-based FFT backend. Initially, the code relied on Intel's Double-Batched FFT (DBFFT) library, which was difficult to install on LUMI's HPE/AMD architecture. The goal was to replace DBFFT with the oneMath library, which acts as a portable layer for vendor-specific mathematical libraries, leveraging ROCm on LUMI.	
Work done	The team began by installing the oneAPI toolkit and the Codeplay plugin for AMD GPU support, followed by the oneMath library, configuring it to use ROCm backends. For CPU compatibility, the DBFFT library was compiled using Level-Zero and OpenCL dependencies. The BigDFT code was then restructured to allow users to select between DBFFT and oneMath backends, with modifications to SYCL routines, Fortran interfaces, and build system macros. Custom compilation scripts (lumi.rc) were developed for both GPU and CPU targets, ensuring proper linking and environment set-up. Performance analysis focused on the Fock miniapp, with Python routines adapted for benchmarking and profiling. A problematic kernel in the Poisson solver was identified and optimized, restoring coalesced memory access for better performance on AMD GPUs. Strong-scaling tests up to 243 nodes demonstrated good parallel efficiency (76.5%), and GPU vs. CPU comparisons showed speed-ups of 7–12x, validating the GPU acceleration approach.	

	 *7 - Figure with the speed-up difference between GPU and CPU version of the code* The Figure above show how the compilation using GPU thanks to EPICURE support improve the timings of the project runs.
Improvements provided	The project successfully ported BigDFT to LUMI's AMD GPUs using oneMath, avoiding dependency on Intel-specific libraries. The restructured code now supports both DBFFT and oneMath backends. By using oneMath instead of DBFFT, the speed-up of using GPUs increased from 7x to 12x when compared to a CPU-only run.
Conclusions	The project demonstrated that SYCL and oneMath enable portable, high-performance GPU acceleration for BigDFT on AMD hardware, despite initial challenges with proprietary extensions. Also, we also showed that kernel performance is not necessarily transferable across platforms.

Project 92	**EHPC-DEV-2024D05-021**	IT4I@VSB
Machines: Karolina		Collaborators: None
Project goals	The project aimed to establish a robust, interactive development environment for machine learning workflows on the Karolina HPC system. The primary objectives were to set up a Jupyter Notebook workflow with port forwarding for direct access to compute resources, configure and optimize environments for Ollama and BitsAndBytes, and enable efficient model training—both for smaller models on single nodes and for production-scale models using multi-GPU and multi-nodes.	
Work done	The team successfully established a Jupyter Notebook workflow with port forwarding, allowing interactive development directly on Karolina's compute nodes. For Ollama, a Conda environment was created and debugged, resolving multiple dependency issues to ensure smooth operation. A dedicated Python environment for BitsAndBytes was developed, addressing version and compatibility challenges to support model quantization and experimentation. Training runs were conducted on single nodes using available GPU resources, completing experiments with smaller models. However, some attempts to train	

	production-size models hit GPU memory limitations, preventing successful execution. While multi-node distributed training was planned, it was not implemented due to time constraints on the PI's side.
Improvements provided	The project delivered a functional, interactive development environment on Karolina, enabling seamless access to HPC resources via Jupyter Notebook.
Conclusions	The project underscored the importance of early resource assessment and multi-node planning when working with production-scale models, as single-node GPU memory constraints can quickly become a bottleneck. In addition, new tools (Ollama, BitsAndBytes) often have complex dependencies that require careful set-up and validation. Maintaining reproducible environments accelerates future onboarding.

Project 90	**EHPC-AI-2024A04-095**	CINECA
Machines: Discoverer		Collaborators: INESC TEC
Project goals	The project aimed to provide a stable GASNet installation, which initially showed severe delays and frequent crashes.	
Work done	We started by downloading and compiling UPC++ with MPI support via OpenMPI and then downloaded and set up GASNet. Some of the issues with GASNet were due to the system checking for pinnable memory, which could be disabled via environment variables (setting up variables like GASNET_PHYSMEM_MAX). From here, setting up the GPU version of the code was straightforward, and most of the issues were again avoided by an appropriate environment set-up.	
Improvements provided	We provided a GASNet installation with both CPU-only and GPU support, while also providing guidance on how to set up the environment variables for simulations.	
Conclusions	This project highlighted the necessity of appropriate environment variables to modulate things like memory usage. We stored all documentation on https://opencode.it4i.eu/epicure/project-90-ehpc-ai-2024a04-095	

Project 89	**EHPC-DEV-2024D11-037**	CINECA
Machines: Leonardo		Collaborators: None
Project goals	Provide an installation of BFEX for further optimizations to be performed as part of WP3.	
Work done	We created a Conda environment containing all the requirements for the BFEX workflow (such as NumPy or Pandas). We also added AmberTools22 to the same environment, since it is also used to perform a few of the analysis in the workflow. While straightforward, the environment set-up contained several steps, so everything was recorded in a script in case re-installation was needed.	
Improvements provided	We provided the required set-up to perform further optimization under WP3.	
Conclusions	This was a straightforward Conda environment set-up required for future work.	

Project 88	**EHPC-AI-2024A04-062**	IT4I@VSB
Machines: Leonardo		Collaborators: CINECA
Project goals	The project aimed to enable large-scale AI workloads on the Leonardo supercomputer by stabilizing distributed training infrastructure, adapting machine learning workflows to HPC constraints, and establishing robust, offline-compatible experiment tracking.	
Work done	The team reused and stabilized a previously deployed Ray cluster for distributed inference and training, ensuring seamless integration with Slurm for multi-node usage. The vLLM serving system was successfully deployed over Ray, resolving configuration issues and confirming functionality in distributed settings. For large model fine-tuning, the team enabled training of a 70B parameter model using Hugging Face Accelerate with Fully Sharded Data Parallelism (FSDP), leveraging environment variables for CPU RAM-efficient loading and memory scaling. The training ran successfully across 4 nodes (16 GPUs). A reusable Slurm job script was developed, incorporating optimized resource requests, environment set-up, and exclusion of unstable nodes. To address network isolation, an offline logging solution for Weights & Biases (WandB) was implemented, using WANDB_MODE=offline during job execution and WandB sync on the login node post-execution. However, WandB's hyperparameter sweep functionality remained incompatible due to outbound access limitations. The team also produced a comprehensive best practice guide for AI on Leonardo, covering environment set-up, Slurm configurations, Ray and vLLM deployment, offline experiment tracking, and troubleshooting tips.	
Improvements provided	The project delivered a stable, multi-node distributed training and inference infrastructure on Leonardo, enabling successful execution of large-scale AI workloads. The integration of Ray and vLLM, combined with optimized Slurm job scripts, provided a robust foundation for distributed tasks. The offline WandB workflow ensured experiment tracking despite network constraints.	
Conclusions	The project highlighted that memory efficiency, careful configuration and environment tuning are critical for training at scale with FSDP for large models. Adapting toolchains to HPC constraints—such as using WandB in offline mode—proved essential, though some advanced features remained limited. This project resulted in a Best Practice Guide for AI on Leonardo (https://eurohpcsupport.eu/best-practice/best-practice-guide-to-running-ai-workloads-on-leonardo-cineca/), covering the most important aspects of running AI workloads.	

Project 86	**EHPC-AI-2024A04-084**	BSC
Machines: MareNostrum 5		Collaborators: BSC
Project goals	The project is the continuation of a Benchmark activity already completed (see Project 66 on D2.1). The work is focused on benchmarking and scaling up LLM models in MareNosturm 5 ACC partition.	
Work done	As part of our ongoing efforts to optimize and improve the performance of our AI-related tools and services, we have conducted a comprehensive benchmarking exercise to evaluate the scalability of the vLLM frameworks. The primary objective of this benchmark was to assess the framework's ability to execute efficiently across multiple nodes. We observed excellent scalability across several GPU nodes (tests were done	

	from 4 to 32 nodes). We documented our finding in documentation available for the general users of the cluster, with examples available in the machine already primed for direct usage. https://opencode.it4i.eu/epicure/epicure-sw/-/tree/main/scripts/jobscripts/LLM/66-and-86
Improvements provided	This project showed the feasibility to run large language models on MareNostrum ACC using up to 128 GPUs which showed enough for the user requirements. In addition, it resulted in documentation available to all users which greatly reduces the set-up time for experiments.
Conclusions	Our benchmarking exercise has demonstrated the scalability of vLLM. We are committed to continuing our support for users and providing resources to facilitate the adoption of our AI tools and services.

Project 85	**EHPC-DEV-2024D11-072**	IT4I@VSB
Machines: Karolina, LUMI		Collaborators: CSC
Project goals	Provide installation, optimization and scalability improvements for a traffic simulator that relies on evkit.	
Work done	LUMI: evkit was installed using GCC. Out-of-the-box, it only worked for GCC 12, so minor adaptations were required to make it work with GCC 13, mainly due to Rust dependencies. Help was provided on setting up job scripts, and the installation was tested in single-node and multi-node configurations. Karolina: Installation on Karolina followed the same lines as in LUMI, but only single-node performance was tested by the team.	
Improvements provided	Provided an installation of evkit, along with scripts to use the traffic simulator on both machines.	
Conclusions	We successfully provided the installation in the two systems, which then paved the way to perform optimizations under WP3.	

Project 79	**EHPC-DEV-2024D09-067**	LUXPROVIDE SA
Machines: MeluXina		Collaborators: None
Project goals	The project aimed to address performance and convergence issues in the current genetic algorithm (GA) implementation, which was reported to improve solutions too slowly for large problems and to be computationally inefficient.	
Work done	The team focused on revising the GA's solution representation to reduce the search space and simplify the algorithm, eliminating the need for solution "repair" required by the previous approach. Genetic operators were updated to align with the new representation. These changes resulted in a 55% improvement in median solution fitness, significantly accelerating convergence. Throughout the process, the team maintained the GA's generality and simplicity, ensuring the solution remained applicable to a broad range of problems.	
Improvements provided	We achieved a 55% improvement in median solution fitness, accelerating convergence significantly.	
Conclusions	The collaboration was highly effective, setting up a good starting point to perform further code optimizations under WP3.	

Project 78	**EHPC-DEV-2024D09-054**	INESC TEC
Machines: Deucalion, Discoverer, Leonardo, MareNostrum 5, MeluXina, Vega		Collaborators: CINECA, BSC, SofiaTech, LUXPROVIDE SA, JSI, IZUM
Project goals	Providing code porting and optimization support for EuroHPC systems by ensuring the successful deployment and efficient use of state-of-the-art performance analysis and tuning tools across multiple supercomputing sites.	
Work done	We performed the installation of Score-P, Scalasca, Extrae, MERIC, DLB, and MAQAO in all the selected machines. Minor code modifications were also applied where necessary, such as adding compiler pragmas to improve vectorization or adjusting optimization levels to mitigate compiler-related issues.	
Improvements provided	We provided a suite of performance analysis tools on all the required systems, enabling users to better analyse and benchmark their codes.	
Conclusions	While straightforward, it is hard to underestimate the importance of appropriate linking and minor code edits to get the best performance out of these tools.	

Project 70	**EHPC-BEN-2024B09-047**	LUXPROVIDE SA
Machines: MeluXina		Collaborators: None
Project goals	The main goal was to port and optimize FVAP, an AI-driven virtual assistant for portfolio management based on a Turkish-language LLM fine-tuned with Gemma2, ensuring accurate financial reasoning, natural dialogue, and scalable performance on GPU-accelerated supercomputing systems. A key objective was to achieve efficient single- and multi-node training and inference suitable for deployment across financial institutions.	
Work done	Work done included profiling the base implementation to identify CPU dataloader bottlenecks, GPU idle times caused by optimizer offloading, and inefficient distributed synchronization due to small gradient buckets. Successive optimization stages addressed these issues by parallelizing data loading, tuning DDP bucket sizes, introducing fused GPU optimizers, batching gradients, reducing batch sizes where appropriate, and enabling mixed-precision training. Multi-node execution on the MeluXina GPU system was configured using Hugging Face Accelerate and NCCL, with GPU Direct RDMA, cross-NIC communication, optimized buffer sizes, and correct Slurm-based rank assignment to ensure stable and efficient scaling.	
Improvements provided	The optimizations performed resulted in a two to three-fold increase in the training speed.	
Conclusions	Improvements provided resulted in continuous and balanced GPU utilization, elimination of CPU-side stalls, tighter and more predictable training loops across DDP ranks, and significantly reduced communication latency. NCCL bandwidth was better exploited through fewer and larger collective operations while fused optimizers and mixed precision removed costly CPU-GPU transfers and accelerated both training and inference.	

Project 62	**EHPC-DEV-2024D10-054**	LUXPROVIDE SA
Machines: MeluXina		Collaborators: FZJ
Project goals	Improving the parallel performance and scalability of a deal.II-based atmospheric flow solver on the MeluXina supercomputer. Furthermore, we aimed to distinguish between performance limitations originating from the user implementation versus those originating from the deal.II library.	
Work done	Work done included profiling the MPI-parallel C++ code, identifying clear computational hotspots, and discussing the findings with the PI. Job submission parameters were analysed and optimized, showing that using full nodes with 128 MPI tasks and one CPU per task, together with properly setting DEAL_II_NUM_THREADS and OMP_NUM_THREADS to match SLURM_CPUS_PER_TASK, yielded the best scalability. At the PI's request, a focused investigation was conducted on the deal.II built-in compute_diagonal() function, comparing it with a custom implementation using the MAQAO performance analysis tool.	
Improvements provided	Improvements stemmed from identifying suboptimal vectorization in the built-in compute_diagonal() routine, which exhibited a vectorization ratio of 30% or less, in contrast to the custom implementation achieving full vectorization.	
Conclusions	Careful profiling and correct runtime configuration are essential for achieving good scalability. The study showed that performance differences can arise not only from user code but also from library-level implementation details.	

Project 34	**EHPC-AI-2024A02-043**	BSC
Machines: MareNostrum 5		Collaborators: None
Project goals	Enable reliable multi-node training of Python deep learning workloads using torchrun on Slurm-managed systems and to improve the monitoring and analysis of training metrics through appropriate open-source tooling.	
Work done	Work done included the preparation and delivery of two Slurm job scripts for launching multi-node training jobs, along with clear instructions on their usage, node configuration, dataset path set-up, and the integration of a GPU monitoring task running alongside training. In parallel, guidance was provided on improving training observability by integrating the WandB framework, covering installation, local server set-up via Docker, authentication, offline log synchronization, and visualization of training and validation metrics.	
Improvements provided	We provided a streamlined and reproducible workflow for distributed training, allowing users to easily scale jobs across multiple nodes with torchrun. Training runs became more transparent through continuous GPU monitoring and structured metric logging, enabling detailed post-run analysis beyond standard Slurm output files.	
Conclusions	The adopted approach improves usability for multi-node Deep Learning workflows, providing better insight on model behaviour and training.	

3. EPICURE Benchmarking

3.1 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 environments.

3.2 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.

The list of applications included currently in the benchmark suite are described and explained in the previous D2.1 resubmitted in January 2026.

3.3 Implementation for M12 to M24

To date, the EPICURE Benchmark Suite has been deployed and tested on five EuroHPC systems. During the lifetime of the project, it will be further extended to additional sites:

- **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.

[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 monitoring efficiencies, https://pm.bsc.es/ftp/dlb/doc/user-guide/how_to_run_talp.html
[6] MareNostrum 5 Supercomputer, https://www.bsc.es/es/marenostrum/marenostrum-5
[7] Leonardo Supercomputer, https://leonardo-supercomputer.cineca.eu

- **Deucalion**[8] (Portugal): A petascale system with both CPU (ARM and x86) and GPU partitions, supporting advanced research across multiple scientific domains.
- **Vega**[9] (Slovenia): A petascale EuroHPC supercomputer located in Maribor, equipped with both CPU and GPU partitions.
- **LUMI**[10] (Finland): A EuroHPC pre-exascale supercomputer hosted at CSC's data centre in Kajaani, offering CPU and GPU partitions for large-scale scientific workloads.

Initial benchmarking and performance analysis indicate 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 and system 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.

job_details:

Job_ID	inp_filename	nodes	tasks_per_node	num_tasks	threads_per_task	Self_Time_Average	WallTime_Result	Start_Time	Parallel_Efficiency
12804345	H2O-DFT-LS	1	2	2	56	0.212	426.187	2024-12-09 13:59:47	0.58
12804346	H2O-DFT-LS	1	4	4	28	0.114	372.587	2024-12-09 13:59:47	0.41
12804347	H2O-DFT-LS	1	7	7	16	0.108	355.103	2024-12-09 13:59:47	0.43
12804348	H2O-DFT-LS	1	8	8	14	0.071	338.433	2024-12-09 13:59:47	0.49
12804349	H2O-DFT-LS	1	16	16	7	0.038	306.317	2024-12-09 13:59:47	0.61
12804350	H2O-DFT-LS	1	28	28	4	0.038	359.302	2024-12-09 13:59:47	0.58
12804351	H2O-DFT-LS	1	56	56	2	0.029	370.501	2024-12-09 14:00:11	0.63
12804352	H2O-DFT-LS	1	112	112	1	0.045	446.479	2024-12-09 14:00:30	0.65

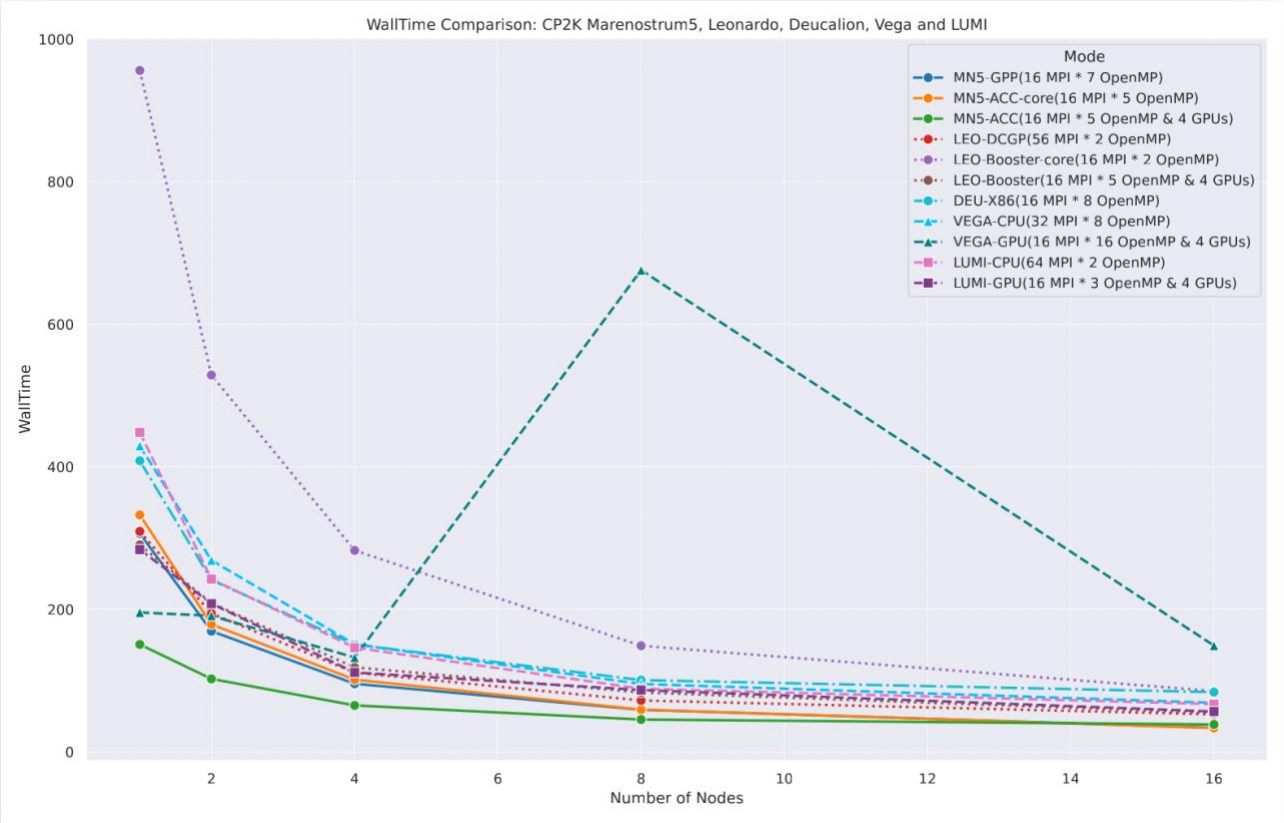
performance_metrics:

Job_ID	Elapsed_Time	Parallel_Efficiency	Communication_Efficiency	Load_Balance	LB_in	LB_out	Useful_Instructions	Total_Useful_Cycle	Avg_IPC	Total_Useful_Time
12804345	426.38	0.58	0.58	0.99	0.99	1.0	1701566788770	1062978605497	1.6	493721432362
12804346	372.8	0.41	0.42	0.98	0.98	1.0	1963538756170	1182602240260	1.66	615454528053
12804347	355.33	0.43	0.49	0.88	0.88	1.0	3216140027682	2333148957757	1.38	1072643713866
12804348	338.65	0.49	0.52	0.94	0.94	1.0	4129063840689	2695288500020	1.53	1331386045115
12804349	306.55	0.61	0.7	0.87	0.87	1.0	9152171392098	6362993560917	1.44	2999188220081
12804350	359.55	0.58	0.66	0.88	0.88	1.0	16874043280964	13205524458516	1.28	5849381510236
12804351	370.81	0.63	0.76	0.83	0.83	1.0	36084489431616	29824646107243	1.21	13054716293632
12804352		0.65	0.79	0.82	0.82	1.0	77257694901622	74888008939112	1.03	32302356803234

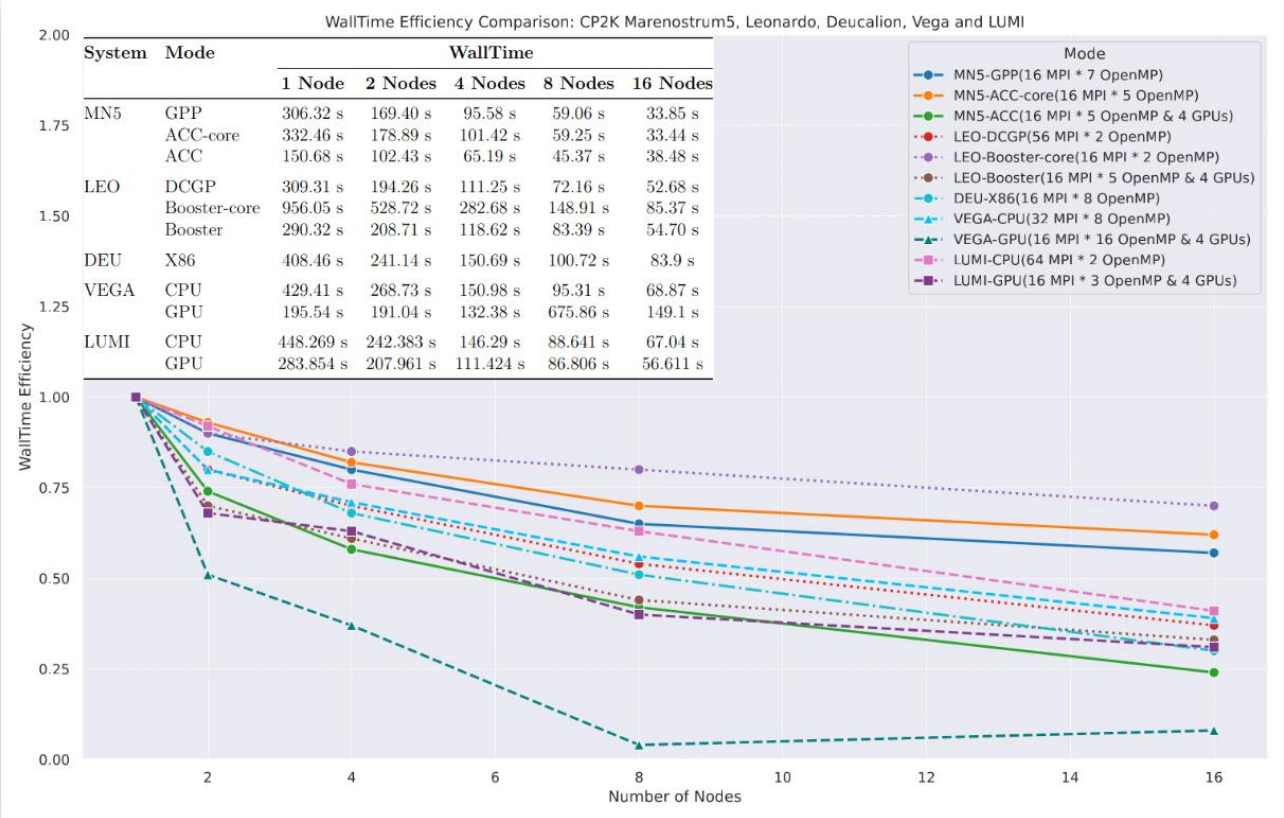
8 - Figure with the metrics generated from JUBE for CP2K on 1 node.

The JUBE output illustrates automated benchmark result collection, with metrics presented in structured tables.

[8] Deucalion Supercomputer, https://docs.macc.fccn.pt/
[9] Vega Supercomputer, https://en-vegadocs.vega.izum.si/introduction/
[10] LUMI Supercomputer, https://docs.lumi-supercomputer.eu/



9 - Figure with CP2K results from the benchmark suite running on different supercomputers.



System	Mode	WallTime				
		1 Node	2 Nodes	4 Nodes	8 Nodes	16 Nodes
MN5	GPP	306.32 s	169.40 s	95.58 s	59.06 s	33.85 s
	ACC-core	332.46 s	178.89 s	101.42 s	59.25 s	33.44 s
	ACC	150.68 s	102.43 s	65.19 s	45.37 s	38.48 s
LEO	DCGP	309.31 s	194.26 s	111.25 s	72.16 s	52.68 s
	Booster-core	956.05 s	528.72 s	282.68 s	148.91 s	85.37 s
	Booster	290.32 s	208.71 s	118.62 s	83.39 s	54.70 s
DEU	X86	408.46 s	241.14 s	150.69 s	100.72 s	83.9 s
VEGA	CPU	429.41 s	268.73 s	150.98 s	95.31 s	68.87 s
	GPU	195.54 s	191.04 s	132.38 s	675.86 s	149.1 s
LUMI	CPU	448.269 s	242.383 s	146.29 s	88.641 s	67.04 s
	GPU	283.854 s	207.961 s	111.424 s	86.806 s	56.611 s

10 - Figure with CP2K wall time efficiency on different supercomputers.

The wall time and wall time efficiency graphs highlight performance differences across sites for each benchmark.

Overall, the EPICURE Benchmark Suite has proven to be an effective tool for automating software installation, benchmarking, and performance analysis in HPC systems. It enables the EPICURE team to detect possible cross-site optimization, architecture efficiency for different applications and detect difference between sites at configuration level, which may significantly impact application performance.

3.4 Future work

The next step is to deploy the EPICURE Benchmark Suite on additional EuroHPC systems, including Discoverer, JUPITER, Karolina and MeluXina, in order to obtain a comprehensive view of performance across all participating sites. This broader deployment will enable cross-system validation and help identify architectural trends, tuning opportunities and portability challenges across diverse hardware platforms.

In parallel, future work will concentrate on extending the benchmark suite to include additional highly scalable applications, particularly those that are relevant to exascale computing. Applications currently under consideration include:

- iPIC3D (plasma physics),
- URANOS (CFD),
- Vlasiator (space/plasma).

These three codes have already demonstrated very good scalability on large numbers of nodes, and some of them support both GPU and CPU execution, which represents a clear advantage for inclusion in the benchmark suite. Over the next twelve months, the main focus will be to define the final set of applications to include, identify suitable input datasets, and complete the deployment of the benchmark suite and framework across the remaining sites. Additional effort will be dedicated to improving automation documentation, so that the suite can be used both by system operators and by application developers.

In the longer term, the plan is to introduce performance-analysis workflows to better understand bottlenecks, and to track performance evolution across software and hardware updates. This will help ensure that the EPICURE Benchmark Suite remains relevant as new architectures, programming models and optimization strategies emerge within the EuroHPC ecosystem.

4. Best Practices Guidelines for EuroHPC supercomputers and technical reports

4.1 Introduction and status

During the first 24 months, the work focused on defining potential topics for the 2025/26 user guidelines and on publishing the first best-practice guidelines at Month 18, which concentrated on power-consumption measurement on EuroHPC systems.

In parallel, a new structure was established for the EPICURE teams to publish technical documentation, scripts and software through the project GitHub repository (https://opencode.it4i.eu/epicure/epicure-sw).

The repository was organised as follows:

- compilation: Build and compilation scripts not linked to tools such as EasyBuild or Spack.
- containers: Docker and Singularity definitions; larger images are stored in a separate repository (https://opencode.it4i.eu/epicure/container-images-repo).
- recipes: Installation logs, reports, and scripts generated by EasyBuild, Spack, or other automated tools.
- scripts: Utility scripts for job execution, monitoring, performance testing, and related workflows.
- support-tools: Additional utilities to assist development, deployment, and auxiliary tasks.

This organization allows users, system administrators, and support teams to easily access and reuse build procedures, automation scripts, and lightweight monitoring tools developed by the project.

The repository has already been populated with several examples. Over the next twelve months, it is expected to become an important resource for the project, providing the wider HPC community with access to build procedures, scripts and lightweight monitoring tools that may be useful for users, system administrators and support teams.

4.2 Best Practices Guidelines

The first best practice guideline was published in July 2025: “The Best Practice Guide on Power Consumption Measurements in EuroHPC Systems”, available online[11].

This guide provides practical, hands-on instructions for users to access, collect, interpret, and analyse power and energy consumption data on active EuroHPC supercomputers. It details the various measurement tools and interfaces available across different systems and includes example job scripts and workflows to help users monitor the energy footprint of their applications.

The guide begins with an introduction to power monitoring, followed by a description of the benchmark suite used throughout the document. This suite includes NAMD, GROMACS, and CP2K, along with their corresponding input files. All three applications have been executed on both CPU and GPU partitions across all EuroHPC sites. Using widely known applications across multiple sites enables a comparative energy analysis across different hardware architectures.

[11]The Best Practice Guide on Power Consumption Measurements in EuroHPC Systems v1 https://epicure-hpc.eu/wp-content/uploads/2025/07/EPICURE-BEST-PRACTICE-GUIDE-Power-measurements-in-EuroHPC-machines_v1.0.pdf

The document then provides an overview of the EuroHPC systems, including LUMI, Leonardo, MareNostrum 5, MeluXina, Karolina, Discoverer, Vega, Deucalion, and JUPITER, explaining how power and energy data can be collected on each system.

A central section of the guide focuses on the tools and methods for power measurements, including:

- integrations with Slurm accounting, EAR (Energy Aware Runtime), MERIC runtime system, and COUNTDOWN,
- CPU measurements via RAPL interfaces,
- GPU measurements via NVML for NVIDIA GPUs,
- platform-specific dashboards and utilities that provide power metrics to users.

The guide also provides practical examples, including job scripts demonstrating how to run benchmarks with power measurement enabled. Additionally, it includes visualizations such as performance and energy graphs, normalized energy metrics (e.g., energy per nanosecond per day), and heat maps that correlate energy use with performance outcomes.

An update to this guideline was made in January 2026, incorporating the following revisions:

- Review of outliers: Performance and energy values exhibiting unusual behaviour were re-examined. For these cases, the team repeated the runs to determine whether the anomalies were reproducible or caused by temporary issues with the power monitoring tools.
- Enhanced graph explanations: Additional details were added to the graphs to clarify possible scenarios leading to deviations from normal behaviour.
- Improved conclusions: Final conclusions for each application were refined with additional insights and information.

The updated version of the “The Best Practice Guide on Power Consumption Measurements in EuroHPC Systems” is available on the European HPC Application Support Portal[12].

For the second guideline, the focus has been set on AI workflow execution and optimization, following this plan:

- best practices for running AI workflows across different EuroHPC architectures (AMD, NVIDIA, etc.),
- strategies for optimizing parallel AI workflows and addressing common challenges,
- additional considerations including installation, execution, job configuration, and I/O for AI workflows.

This second guideline is scheduled for publication in Month 28 of the project, which is at the end of May 2026.

[12] The Best Practice Guide on Power Consumption Measurements in EuroHPC Systems v2 https://eurohpcsupport.eu/wp-content/uploads/2025/07/EPICURE-BEST-PRACTICE-GUIDE-Power-Consumption-Measurements-in-EuroHPC-Systems.pdf

5. Conclusions

The second year of the project (from M12 to M24) focused on stabilizing operations, improving coordination with teams, enhancing response times for projects, strengthening collaboration with EuroHPC-JU, and laying the groundwork to expand EPICURE's benchmarking, technical reports, and user guidelines activities.

While the primary efforts centred on providing direct support to projects, the benchmarking and best practice guides have proven to be valuable resources, significantly increasing the benefits for the HPC community by offering tools and guidance useful for their day-to-day work with the EuroHPC systems.

During the second year, EPICURE demonstrated the ability to support over 57 projects, delivering high-quality technical assistance, reducing the set-up burden for projects, and promoting HPC usage among communities with limited HPC experience.

Overall, the last 12 months highlighted EPICURE's reliability in project support and its potential for future initiatives, such as adopting new benchmarks and publishing technical reports and best practice guides. These outputs provide practical value not only for the general HPC community but also specifically for EuroHPC systems, benefiting users, system administrators, user support teams, and application developers alike.