D3.2 Scalability, Optimization and Co-Design Activities

Date: April 30, 2025

Document Identification			
Status	Final	Due Date	30/04/2025
Version	1.0	Submission Date	30/04/2025

Related WP	WP3	Document Reference	D3.2
Related Deliverable(s)	D3.1, D3.4, D3.5	Dissemination Level (*)	PU
Lead Participant	ICCS	Lead Author	Konstantinos Nikas, Petros Anastasiadis (ICCS)
Contributors	MTG, PSNC, SZE, UNISTRA, FAU	Reviewers	Zoltán Horváth (SZE)
			Philippe Pinçon (UNISTRA)

Keywords:

High Performance Computing (HPC), benchmarking, scalability, optimization, application bottlenecks, performance analysis

Document Information

List of Contributors	
Name	Partner
Petros Anastasiadis	ICCS
Kostis Nikas	ICCS
Michal Kulczewski	PSNC
Wojciech Szeliga	PSNC
Leydi Laura	MTG
Luis Torres	MTG
Angela Rivera	MTG
David Caballero	MTG
Javier Cladellas	UNISTRA
Vincent Chabannes	UNISTRA
Christophe Prud'homme	UNISTRA
László Környei	SZE
Michael Zikeli	FAU

Document History			
Version	Date	Change editors	Changes
0.1	25/02/2025	ICCS	Initial version of the document - ToC
0.2	07/03/2025	Marcin Lawenda (PSNC), Rahil Doshi (FAU)	ToC, timeline and responsibilities approved
0.3	24/03/2025	ICCS, MTG	ICCS - MTG contributions for WF pilot
0.4	12/04/2025	ICCS	Combined ICCS & MTG WF contributions
0.5	12/04/2025	ICCS	Integrated FAU-MTW contributions
0.6	13/04/2025	PSNC, ICCS	Integrated PSNC-RES contributions
0.7	13/04/2025	UNISTRA, SZE, ICCS	Integrated UNISTRA-UB and SZE-UAP contributions
0.8	16/04/2025	ICCS	Draft for internal review
0.9	24/04/2025	ICCS	Integrated Internal review comments
0.95	29/04/2025	Rahil Doshi	Quality assurance check

1.0	30/04/2025	Marcin Lawenda	Final check and improvements

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Petros Anastasiadis (ICCS)	29/04/2025
Quality manager	Rahil Doshi (FAU)	29/04/2025
Project Coordinator	Marcin Lawenda (PSNC)	30/04/2025

Table of Contents

List of Tables

List of Figures

List of Acronyms

Abbreviation / acronym	Description
EC	European Commission
CFD	Computational Fluid Dynamics
CoE	Centre of Excellence
CPU	Central Processing Unit
Dx.y	Deliverable number y belonging to WP x
DXQY	X represents the model's dimensions; Y represents the number of PDFs with which the lattice cell is discretized
EESSI	European Environment for Scientific Software Installations
EULAG	Eulerian/semi-Lagrangian fluid solver
FPGA	Field-programmable gate array
GFS	Global Forecasting System
GIS	Geographic Information System
GPU	Graphics Processing Unit
GRIB	GRIdded Binary or General Regularly-distributed Information in Binary form
HPC	High Performance Computing
I/O	Input/Output
KPI	Key Performance Indicator
LB	Lattice Boltzmann
LBM	Lattice Boltzmann Method
LOD	Level of Detail
MPI	Message Passing Interface
MRT	Multiple Relaxation Time
MTW	Material Transport in Water
PDF	Particle Distribution Functions
RDM	Research Data Management
RES	Renewable Energy Sources

SIF	Singularity Image Format
SRT	Single Relaxation Time
UAP	Urban Air Project
UB	Urban Building
WF	Wildfires
WP	Work Package
WPS	WRF Pre-processing System
WRF	Weather Research and Forecasting Model

Executive Summary

This deliverable presents the current status of scalability, performance optimization, and co-design activities across all HiDALGO2 [1] pilot applications on EuroHPC JU systems. It builds upon D3.1 [2] and focuses on improvements made during the second part of the project (M13-M28). Its main objective is to evaluate how the pilots behave on different HPC architectures, identify performance bottlenecks, and prepare the applications for extreme-scale benchmarking going into the third part of the project.

Compared to D3.1, this deliverable analyses a new pilot (MTW) that has been added to HiDALGO2, extends benchmarks to new systems, introduces larger and more complex scenarios, and explores cross-platform portability through co-design and profiling activities. In addition, profiling tools such as Score-P [3] and Cube [4] have been applied to gain deeper insights into code-level performance and highlight dominant communication or I/O bottlenecks. These insights are now guiding code restructuring and optimization efforts, including potential improvements in MPI load balancing, communication overlap, and memory usage.

In conclusion, D3.2 confirms that most pilots have reached a mature benchmarking phase, scaling well to thousands of cores. However, further work is still required to improve communication efficiency, I/O handling, and portability across heterogeneous systems. These aspects will be the focus of follow-up activities in regards to benchmarking and optimization.

1 Introduction

1.1 Purpose of the document

This deliverable documents the benchmarking, optimization, and co-design activities for HiDALGO2 pilot applications during the second part of the project (M13–M28). It reports the scalability and performance improvement of each pilot on EuroHPC JU systems compared to D3.1, identifies new bottlenecks, and evaluates deployment portability approaches. The results are used to guide future developments optimizations and benchmarking affords for the HIDALGO2 partners.

1.2 Relation to other project work

Deliverable D3.2 compares the performance and scalability of HiDALGO2 pilots in the 16 month period following D3.1, as these are described in deliverables D5.3 "*Research Advancements for the Pilots (M10)*" and D5.4 "*Research Advancements for the Pilots (M23)*", on the project's HPC infrastructure, as defined in deliverable D2.4 "*Infrastructure Provisioning, Workflow Orchestration and Component Integration*". Deliverable D3.2 drives future activities within WP3 (*Exascale Support for Global Challenges*) and WP5 (*Tackling Global Challenges*). It is the second of a series of reports focusing on scalability, optimisation and co-design activities (D3.1 in M12, D3.2 in M28, and D3.3 in M47).

1.3 Structure of the document

This document is structured in 5 major chapters.

- ***Chapter 2*** presents the current EuroHPC deployment status for the project's pilots, as well as challenges of acquiring resources and using EuroHPC JU systems.
- ***Chapter 3*** discusses the benchmarking and optimization progress of each HiDALGO2 pilot, and presents the updated scalability results.
- ***Chapter 4*** presents the status of HiDALGO2 KPIs related to benchmarking and optimisation activities and summarizes all findings.
- ***Chapter 5*** summarizes and concludes this deliverable.

2 Access to EuroHPC JU supercomputers

HiDALGO2 works towards the deployment, benchmarking and optimization of the HiDALGO2 pilots on all EuroHPC JU [5] systems. For this purpose, HiDALGO2 requires simplified access to the appropriate amount of resources on all EuroHPC JU supercomputers.

2.1 Challenges and limitations using EuroHPC JU systems

2.1.1 Limited amount of resources

Since no special access scheme has been provided for European projects, the current standard procedure is still similar to the one described in D3.1, i.e., each HIDALGO2 partner is required to submit separate proposals for resources to the EuroHPC JU. Until now, almost all pilots still leverage the development access scheme, as the regular and extreme access schemes are awarded to already mature codes in terms of scalability.

Unfortunately, in 2024 the amount of resources awarded through the development access calls were significantly reduced, creating a significant challenge for HiDALGO2. More specifically, while in the first year of the project HiDALGO2 pilots were allocated 10000-15000 node hours per CPU partition, now they receive only 3000-4000 node hours; similarly, the allocations of the GPU partitions are awarded between 40% and 80% of the resources that were allocated in the first year of the project. The limited resources create a significant challenge, as all HiDALGO2 pilots are expected to be under continuous development and improvement during the lifetime of the project. Hence, even though our current efforts demand significantly more computational resources for testing, validation and profiling than the first year, they have been actually performed with a more constrained budget.

Additionally, during the reporting period, several hardware issues (especially in newly added systems), such as nodes crashing during execution or network issues between the nodes, caused draining of the awarded resources. Consequently, pilot executions for larger domains and fine-grained resolution requiring many CPU cores easily consumed the resources allocated via development calls, leading to continuously reapplying for additional resources. For example, the execution of the WF pilot for a *single benchmark scenario* drained all the resources of a single development call on the Leonardo DCGP partition; the same was true for the profiling of WF using Score-P on Karolina.

Finally, some of the systems (Karolina GPU, MeluXina CPU) experienced oversubscription of their resources for some period of time (around 6 months), leading

to them being entirely unavailable and not awarding resources to applicants during this time.

2.1.2 System maintenance and large queue times

Another challenge faced during the reporting period was that many systems experienced significant downtimes due to scheduled maintenance or unexpected emergencies. All systems regularly experienced downtimes lasting several days and even longer in some cases; LUMI was down for almost the entire month of August 2024, and MeluXina for several weeks in September 2024. Additionally, queuing times vary greatly across systems, complicating the process of designing benchmark runs and scheduling them on the systems. For example, the waiting time for larger runs (requesting thousands of CPU cores) typically amounts to a few days for LUMI and VEGA.

2.1.3 System portability issues

A recurring challenge across pilots is the portability of workflows between different HPC systems, primarily due to variations in environment management and job submission practices. While most solver components are portable at the code level, the workflow runners and build environments often face system-specific restrictions. For instance, some pilots are affected by platform constraints such as filesystem limits (e.g., maximum number of files) or restricted external network access required to install dependencies. These issues can prevent consistent and automated setup across systems.

In addition, MPI job submission mechanisms differ between platforms. Some systems require the use of mpirun or mpiexec for efficient parallel execution, while others mandate the use of srun or platform-specific launchers. These inconsistencies complicate workflow portability and reduce reproducibility. As a potential solution, containerised application environments are being explored to decouple software environments from system-specific configurations. While containers offer a promising direction for reducing setup complexity and improving cross-system compatibility, most pilots did not start with containerized applications and this adds extra migration overhead.

2.2 Awarded EuroHPC JU resources (M13-M28)

Table 1 provides the EuroHPC JU systems coverage at the end of the second part of the HiDALGO2 project. Access to systems has been requested in such a way that HiDALGO2 does not focus on a subset of supercomputers and works on as many systems as possible, taking into account the implementations of the pilots. Specifically:

Table 1. Current EuroHPC JU systems coverage matrix. Green cells indicate that access has been awarded, grey cells denote a system that a partner is not planning to request access to, as it is either similar to another partition or not suitable for the pilot, and red cells indicate that an application was made but the system was unavailable due to lack of resources. RES, UAP, UB, WF and MTW columns show the systems requested by each pilot, while the HiDALGO2 column shows the systems requested by ICCS for the HiDALGO2 benchmarking activities in general.

System	Partition	RES	UAP	UB	WF	MTW	HiDALGO2
Discoverer	CPU						
Karolina	CPU						
	GPU						
LUMI	CPU						
	GPU						
Meluxina	CPU						
	GPU						
	FPGA						
Vega	CPU						
	GPU						
Leonardo	CPU						
	GPU						
MareNostrum5	CPU						
	GPU						
Deucalion	CPU-x86						
	CPU-ARM						
	GPU						

- No pilot has an FPGA-based implementation.
- RES, UB and WF are currently implemented only for execution on CPUs.
- UAP uses three different codes. Two of those are implemented solely for execution on multiple CPUs, while the third has also a multi-GPU implementation, targeting NVIDIA GPUs.
- MTW supports NVIDIA and AMD GPU architectures and AMD, Intel and ARM CPUs.

Based on the above, no resources have been requested in the FPGA partition of MeluXina. Also, Deucalion became available relatively recently and even though some pilots requested and have been awarded resources, it has not been the focus of benchmarking for this deliverable.

Table 2. Hardware configuration of CPU partitions used during the reporting period (M13-M28)

	CPU/node	Cores/ node	Mem.	Interconnect
Altair	2x INTEL Xeon 8268	48	192GB	InfiniBand @ 200 Gb/s
Proxima	2 x Intel Xeon Platinum 8480p	112		
LUMI	2x AMD EPYC 7H12	128	256GB	Slingshot-11 @ 200 Gb/s
Discoverer				InfiniBand @ 200 Gb/s
Karolina				
Vega				
MeluXina			512GB	
Leonardo	2 x Intel Xeon Platinum 8480p	112	512GB	
MareNostrum5			256GB	
Deucalion	2x AMD EPYC 7742	128	256GB	
	1x FUJITSU A64FX	48	32GB	InfiniBand @ 100 Gb/s

Table 3. Hardware configuration of GPU partitions used during the reporting period (M13-M28)

	CPU	Mem.	GPU	GPU Mem.
LUMI	1x AMD EPYC 7A53	512 GB	4 AMD MI250x	64 GB HBM2
Karolina	2x AMD EPYC 7763	1 TB	8x NVIDIA A100	40 GB HBM2
Meluxina	2x AMD EPYC 7452	512GB	4x NVIDIA A100	80 GB HBM2
Vega	2x AMD EPYC 7H12			
Leonardo	1x Intel Ice Lake			
Deucalion	2x AMD EPYC 7742			
MareNostrum5	2x Intel Xeon Platinum 8460Y	512GB	4x NVIDIA H100	64 GB HBM2

Table 2 and

Table 3 present the hardware characteristics of all EuroHPC machines currently available via development calls, together with the two PSNC [6] partitions (Altair, Proxima) that are available for HiDALGO2. As the Intel-based systems (Leonardo and MareNostrum5) became available after D3.1, deployment activities during the reporting period focused mainly on them. Further, as most EuroHPC systems feature GPU partitions with NVIDIA A100s, the HiDALGO2 pilots focused mainly on Karolina (8xNVIDIA A100), LUMI-G (4xAMD MI250x) and MareNostrum5 (4xNVIDIA H100) in an attempt to cover different accelerator architectures, taking of course into account the effort required for porting their codes.

3 HiDALGO2 pilots' benchmarking

This section outlines the benchmarking and optimization efforts carried out for the HiDALGO2 pilots. Table 4 provides an overview of the current benchmarking status across the available EuroHPC JU systems. It is noted that Deucalion became accessible only recently and has not yet been the focus of benchmarking. Similarly, MeluXina-GPU and Leonardo-GPU share the same architecture as VEGA-GPU, and for this reason, they have not been prioritized for deployment at this stage.

Table 4. Current benchmarking coverage of the HiDALGO2 project

System	Partition	Pilots	Benchmarked applications
Discoverer	**CPU**	2	UAP-FOAM, UAP-Xyst, UB-Ktirio
Karolina	**CPU**	3	UAP-RedSIM, WF-WRF, UB-Ktirio
	GPU	1	UAP-RedSIM
LUMI	**CPU**	4	RES-EULAG, UAP-FOAM, UAP-Xyst, UAP-RedSIM, WF-WRF, MTW-walBerla
	GPU	1	MTW-walBerla
Meluxina	**CPU**	1	UAP-FOAM, UAP-Xyst, UB-Ktirio
	GPU		*Similar architecture with VEGA-GPU*
	FPGA		*No pilots with FPGA implementations*
Vega	**CPU**	2	WF-WRF, UB-Ktirio
	GPU	1	UAP-RedSIM
Leonardo	**CPU**	2	RES-EULAG, WF-WRF
	GPU		*Similar architecture with VEGA-GPU*
MareNostrum5	**CPU**	1	MTW-walBerla
	GPU	1	MTW-walBerla
Deucalion	**CPU-x86**		*Will be the focus of benchmarking in the next part of the project*
	CPU-ARM		
	GPU		

3.1 Renewable Energy Sources (RES)

The Renewable Energy Sources pilot is dealing with three different scenarios: i) prediction of energy produced by *wind farms*, ii) prediction of energy produced by *photovoltaic (PV)* systems, and iii) prediction of the *damages* to the overhead electrical network. In the previous deliverable (D3.1), the focus of benchmarking was on the *damages* scenario for a small domain representing part of a major city in Poland. The solver scaled well up to a certain point, after which the relatively small problem size per CPU and the communication overheads impaired further scaling. In this deliverable, the scalability performance analysis is extended with: i) a larger domain

with greater resolution for the *damages* case, and ii) a new scenario dedicated to PV energy production *(CBPIO-PV)*.

3.1.1 Pilot progress and updates

The newly introduced scenario, CBPIO-PV, targets the prediction of energy production of photovoltaic systems. While it uses the same solver with the damages scenario (EULAG), there is a difference in setting up the pre-conditioner and the characteristics of some related physics. More specifically, EULAG can account either for the orography (terrain elevation) or for the structure of the buildings, with the latter being used in CBPIO-PV. Additionally, CBPIO-PV uses a new dataset prepared for the sake of this scenario, which depicts part of Poznan, where the PSNC HQ is located, on a roof of which a photovoltaic installation exists. The grid resolution is 2136x2363x46 with horizontal resolution of just 1m, allowing to test EULAG scalability for a far larger domain and with greater horizontal resolution.

For the damages scenario, no code changes were made compared to D3.1.

3.1.2 Co-design activities

The RES pilot has pursued two key co-design directions during this period: i) improving workflow portability across HPC systems, and ii) evaluating opportunities for improving performance through domain decomposition refinements.

With regard to the former, the RES pilot encountered several deployment and portability challenges, especially on the Karolina system, where hardware and networking issues prevented the successful completion of benchmarking within the allocated node-hours. The RES workflow is currently orchestrated through a runner based on Conda, which has proven problematic across HPC systems due to platform-specific constraints, such as file number limits and Conda-related compatibility issues. To address these limitations and improve reproducibility, the team is exploring the definition of system-specific configuration recipes or, alternatively, the delivery of a dockerised version of the RES application. The final target of these efforts is the automated environment setup for benchmarking and production runs, and is still under development.

With regard to the latter, from a performance standpoint, the parallelisation strategy of the RES solver is based on a Cartesian grid decomposition across MPI processes. Current observations suggest that the default CPU-to-MPI mapping applied automatically by the HPC systems provides effective utilisation and good efficiency. However, as outlined in the following sections, further improvements may be achieved by considering network topology during the domain decomposition process. This could help refine the mapping of computational domains to processing elements, potentially enhancing performance in communication-heavy phases of the workflow.

3.1.3 Performance analysis

RES - System configuration

The RES pilot was previously available on LUMI and PSNC's Altair machine. This deliverable focuses on analysing the performance achieved on different CPU vendors, AMD (LUMI-C) and Intel CPUs (Leonardo, Altair, Proxima). Table 5 describes the programming and runtime environments used in each system.

Table 5. Programming & runtime environment for RES benchmarks

	Altair	LUMI	Proxima	Leonardo
Compiler	GNU Fortran v.6.2.0	GNU Fortran v.10.3.0	GNU Fortran v.11.5.0	GNU Fortran v12.2.0
Parallel framework	OpenMPI v.4.1.0	Cray MPICH v.8.1.27	OpenMPI v4.1.7a	OpenMPI v4.1.6
Libraries				
NetCDF-C	4.8.1	4.9.2	4.8.1	4.9.2
NetCDF-Fortran	4.5.3	4.6.1	4.5.3	4.6.1
HDF5-C	1.12.1	1.14.1	1.12.1	1.14.3
HDF5-Fortran	1.12.1	1.14.1	1.12.1	1.14.1
Python	3.10.11	3.9.17	3.10.11	3.10.2

RES - Benchmarking configuration

The performance analysis presented in this deliverable was conducted for four scenarios, the details of which (grid resolution, horizontal spacing, simulated time, and timestep) are presented in Table 6, together with the HPC systems that were used for their execution.

Table 6. Details of RES-EULAG benchmarking scenarios

Scenario	Grid resolution	Horiz. spacing	Simul. time	Timestep	Target systems
R10-dist	320x252x46	10m	1h	0.05	Leonardo
R10	320x252x46	10m	1m	0.05	Leonardo, LUMI, Altair, Proxima
R5	608x472x46	5m	1h	0.05	Leonardo, LUMI, Altair, Proxima
CBPIO-PV-R1	2136x2363x46	1m	1m	0.005	Leonardo

RES - Results & analysis

The R5 scenario was evaluated in D3.1 for up to 85 nodes on Altair and up to 10 on LUMI. Figure 1 depicts the improved scalability for this scenario. It uses up to 128 nodes (6144 cores) on Altair, 64 nodes (8192 cores) on LUMI, 32 nodes (3584 cores) on Proxima and 16 nodes (1792 cores) on Leonardo. For all systems, the code shows relatively good scalability in the initial phase, typically up to around 8-16 nodes. The speedup increases almost linearly with the number of nodes, indicating that there is sufficient computation to utilize more processors effectively. As the number of nodes increases further, the speedup starts to plateau and eventually decrease for some systems, which indicates that the increased communication overhead and the synchronization costs start posing a serious bottleneck to computation.

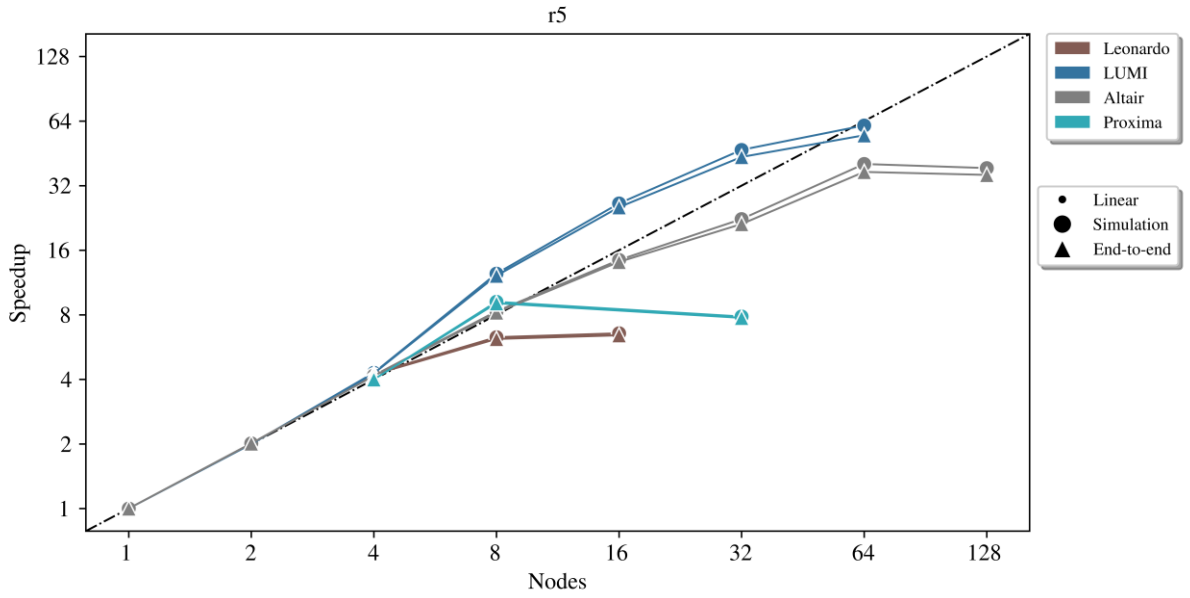


Figure 1. RES-EULAG per node speedup for the R5 scenario

A closer look at system-specific performance reveals major differences in scalability behaviour. Proxima demonstrates good scalability up to 8 nodes, showing trends similar to Leonardo. However, beyond this point, speedup on Proxima plateaus significantly and even declines slightly between 16 and 32 nodes, ultimately achieving the lowest speedup among the four tested systems at higher node counts. Since Leonardo and Proxima are the two systems with the most recent Intel CPUs, this suggests that the RES workload faces architectural constraints on such systems. Further analysis is required to better understand these disparities and identify optimisation opportunities, which is why Leonardo was chosen for RES profiling and further analysis. In contrast, LUMI exhibits the best scalability across all platforms tested. It maintains near-linear speedup up to 32 nodes and continues to scale relatively well up to 64 nodes before gradually plateauing. The super-linear scaling behaviour that was also observed in D3.1. for RES persists, and is attributed to better cache utilisation in higher node counts.

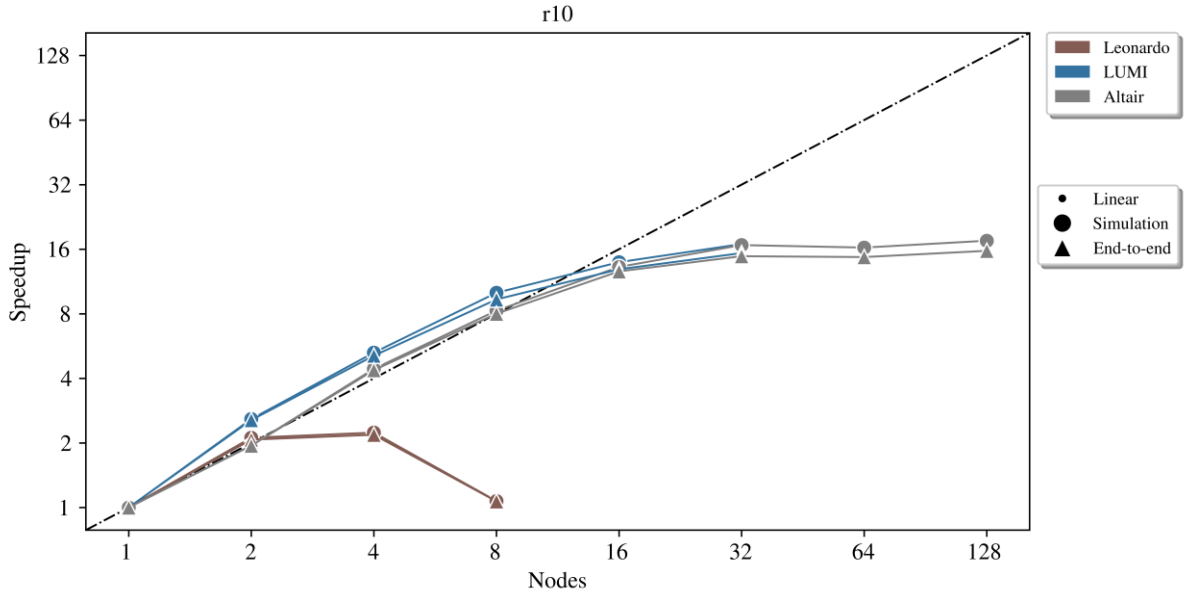


Figure 2. RES-EULAG per node speedup for the R10 scenario

Figure 2 depicts the results for the R10 problem. Similarly to the R5 scenario, all systems demonstrate relatively initial good scalability. Again, LUMI generally shows the best performance and maintains a higher speedup compared to Leonardo and Altair across the higher node counts. On the other hand, saturation appears relatively sooner compared to R5, which is expected since it uses a 3.5x smaller subdomain and therefore is less compute-intensive.

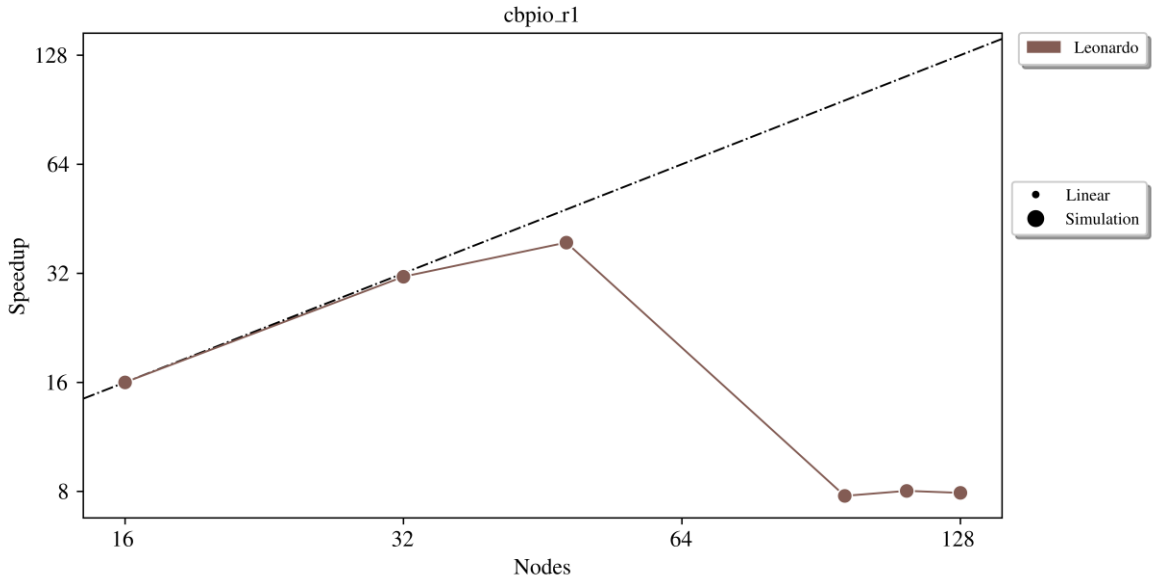


Figure 3. RES-EULAG per node speedup for the CBPIO-RV-R1 fine mesh

CBPIO-RV-R1 is the new scenario introduced in this deliverable by RES, in order to evaluate scalability for very fine meshes using a 1m horizontal resolution. The scalability results for Leonardo system are depicted in Figure 3.

The pilot exhibits relatively good scalability between 16 and 32 nodes, as the speedup nearly doubles as the number of nodes doubles, suggesting efficient utilisation of the increased computational resources. There is a significant performance peak around 48 nodes, achieving a speedup of approximately ~39x, which represents an optimal point where the increased parallelism benefits this specific fine mesh size. However, beyond this peak, the performance collapses drastically. The speedup plummets to approximately 8 at 128 nodes, and adding more nodes after that point does not affect performance. We attribute this dramatic performance drop to inter-node communication overheads, which become increasingly dominant for this very fine mesh at higher node counts, since a significantly larger amount of data needs to be exchanged between neighbouring computational units (which are distributed across different nodes). This puts a much higher strain on Leonardo's interconnect and makes this problem severely communication-bound. Additionally, the finer mesh leads to increased synchronisation between different parts of the computation running on different nodes, further contributing to the problem. Consequently, while load balance is still relatively good locally, the rate at which data can be transferred between the memory systems of different nodes becomes the main bottleneck.

Compared to the previous scalability plots where saturation occurred more gradually, the very fine mesh used here leads to a much more abrupt performance degradation after the peak. This highlights the sensitivity of the application's scalability to the problem size (mesh resolution) and its impact on communication demands. Consequently, for this type of problem, optimisation efforts have to focus on reducing and optimising inter-node communication to achieve better scalability. The size and management of halo cells (overlapping data regions exchanged between neighbours) should be analysed to find whether it can be reduced, or if the update can be performed more efficiently. Another approach is to schedule communication in a way that minimises network contention.

Finally, as Leonardo showed very limited scalability compared to LUMI despite being a more recent system, we used the R10-dist scenario to assess scalability when using all, half, and a quarter of the available CPU cores per node (with unused cores still reserved to isolate performance). Figure 4 depicts execution times for various number of total CPU cores while using all, half, and a quarter of available CPUs within a node.

When using all the CPUs of the allocated nodes, execution time decreases as the number of cores increases up to around 224 cores. However, beyond this point, the execution time starts to increase significantly, indicating poor scalability and diminishing returns from adding more full nodes. On the other hand, using half or a quarter of the available CPUs per node demonstrates better initial scalability compared to the full node configuration. The execution times decrease more consistently as the number of cores increases in the lower total core counts, and using 1/4 of available CPUs is the less time-consuming execution. The increasing execution time for using

all the node's CPUs at higher total core counts suggests that the overhead associated with managing large numbers of cores within fewer nodes becomes a dominant factor.

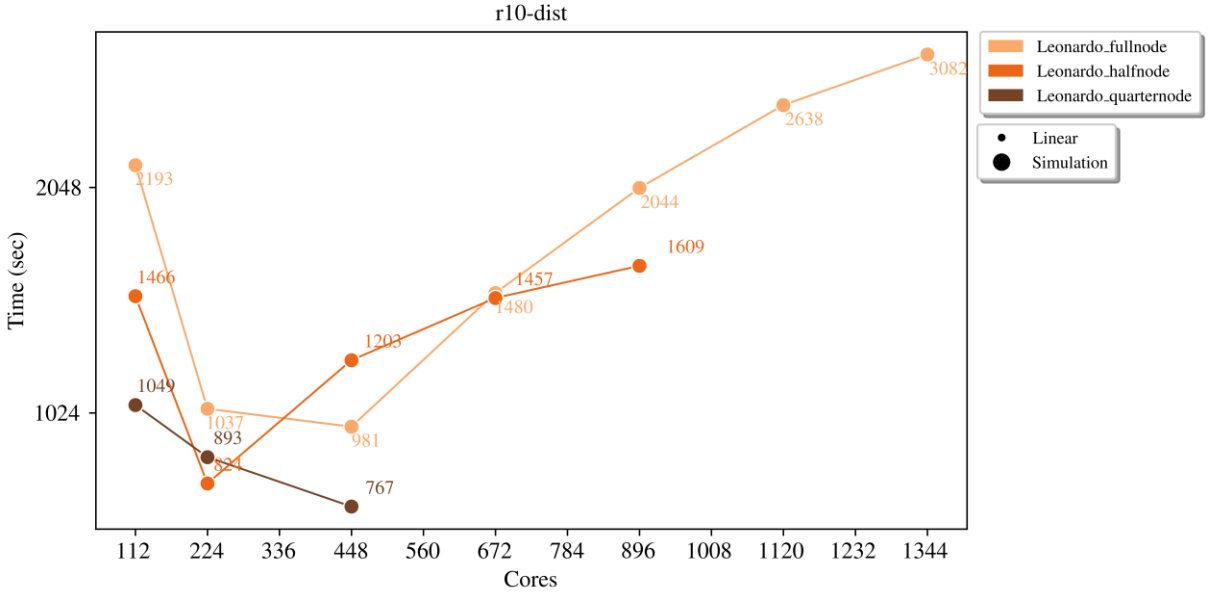


Figure 4. RES-EULAG execution times for the R10-dist scenario when using 25%, 50% and 100% of available CPUS per node

While this issue needs to be further investigated, the potential bottlenecks are:

- *memory contention*, as when using larger number of CPUs per node the available memory bandwidth for each core is limited;
- *cache coherency*, as with larger number of CPUs per node the scope of cache coherence is larger, leading to slower data accesses, and
- *inter-node bandwidth*, as distributing the work across more nodes can allow for better utilisation of the aggregate inter-node bandwidth.

RES - Profiling

To verify previously observed performance issues in the RES pilot, Score-P was used for profiling and event tracing. Since profiling generates large amounts of data, the analysis was limited to the R80 scenario (grid size: 38×30×46, 80 m horizontal resolution), running on 8 CPU cores. This setup was chosen to keep the profiling data manageable while still capturing key performance characteristics. The profiling output, shown in Figure 5, focuses on two main parts of the execution: COM, representing computation, and MPI, representing communication.

The report shows that MPI communication takes up around 46% of the total runtime, and is not overlapped by the computation phase. This confirms that communication delays are a common issue for this setup. The problem appears to be more visible on newer CPU architectures, where faster processors complete the computation more quickly, leading to more time spent waiting on communication. These findings suggest

that improving communication efficiency and reviewing how the domain is split across processors could help improve performance further.

flt	type	max_buf[B]	visits	time[s]	time[%]	time/visit[us]	region
	ALL	1,386,306,222	209,838,745	458.76	100.0	2.19	ALL
	MPI	1,260,895,207	171,251,249	212.95	46.4	1.24	MPI
	USR	68,677,414	21,131,008	20.95	4.6	0.99	USR
	COM	56,733,560	17,456,480	224.86	49.0	12.88	COM
	SCOREP	41	8	0.01	0.0	1129.01	SCOREP

Figure 5. RES timing breakdown - profiling summary

To gain more detailed insights into the RES pilot's performance, the Cube software was used to analyse the profiling traces. Cube focuses on the functions executed and their call paths, providing an overview of the time spent in each function and the distribution of tasks across CPUs (MPI processes), which helps identify hotspots in both computation and communication. The basic profiling results are shown in Figure 6. The left panel presents any performance metrics, which for this case has been configured to execution time. The middle panel displays the execution tree with all called functions and their execution times. The right panel illustrates how each function's work is distributed across MPI processes.

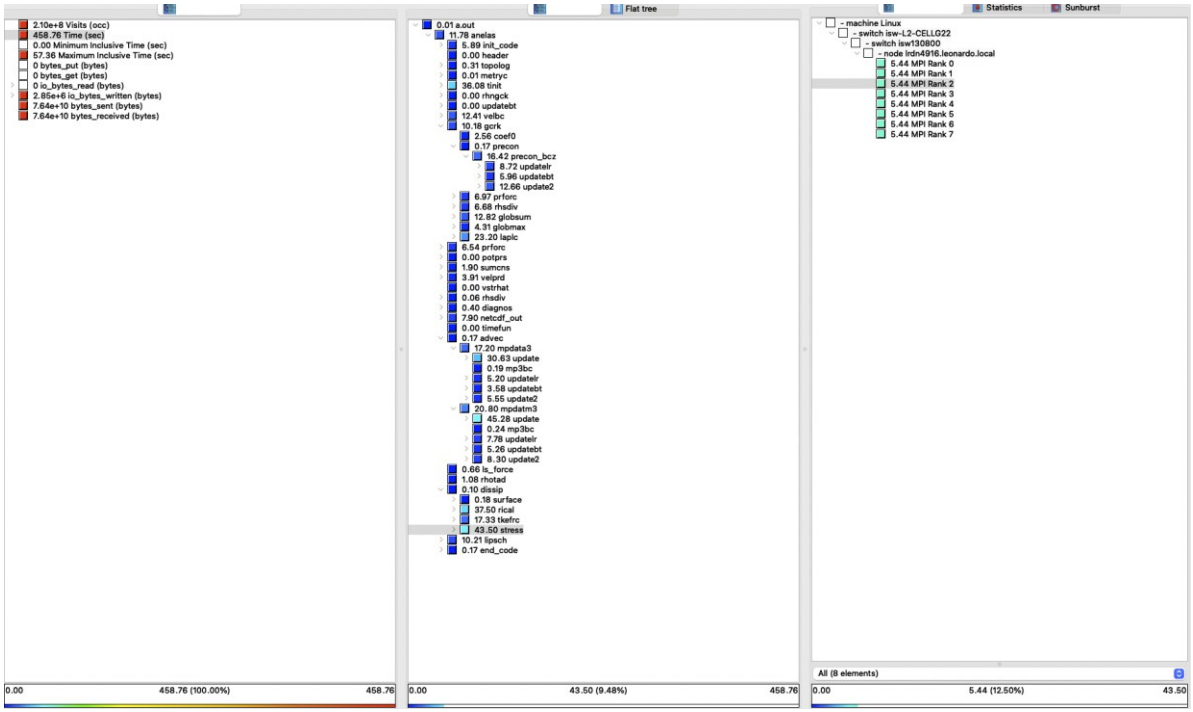


Figure 6. RES Cube basic profiling

In addition, Cube provides several efficiency metrics to help analyse the performance of parallel applications. The ones utilized for RES profiling were:

- *Parallel Efficiency*, which indicates the proportion of time spent on computation versus communication. For example, a value of 80% means that 80% of the total execution time was dedicated to computation, while the remaining 20% was spent on communication.
- *Load Balance Efficiency*, which measures the uniformity of computational workload distribution across MPI processes. It is calculated as the ratio of average computation time to the maximum computation time among all processes, highlighting any imbalance in workload distribution.
- *Communication Efficiency*, which assesses the impact of communication overhead on overall performance. It reflects the proportion of time spent in communication routines relative to computation, identifying inefficiencies due to excessive communication time.

Table 7. RES profiling breakdown of efficiency metrics per function

Function	Time	Parallel Eff.	Load Balance Eff.	Communication Eff.
tinit	8%	0.64 (good)	0.81 (very good)	0.79 (good)
velbc	3%	0.47 (fair)	0.94 (very good)	0.50 (fair)
gcrk	24%	0.62 (good)	0.98 (very good)	0.64 (good)
advec	33%	0.49 (fair)	0.99 (very good)	0.50 (fair)
dissip	21%	0.47 (fair)	0.98 (very good)	0.47 (fair)

Table 7 shows the corresponding metrics for the most time-consuming routines. The load balance is good across all routines, indicating an even distribution of computational work. However, communication efficiency is lower, which impacts the overall parallel efficiency, particularly in the velbc, advec, and dissip functions. Considering both the percentage of total execution time taken by these functions and their parallel efficiency, future optimization efforts for RES should focus particularly on the advec and the dissip functions.

RES – Summary and next steps

Concluding, the RES pilot has identified several areas for performance improvement based on initial profiling and scalability analyses. First, to better understand the application's behaviour under increased parallelism, further profiling will be conducted with a higher number of MPI processes. This will help assess how efficiency metrics evolve as the process count increases. In addition to this, the analysis will explore metrics related to communication efficiency, such as serialization and transfer efficiency. Serialization efficiency will reveal which processes experience delays waiting for others, while transfer efficiency will provide insights into whether overlapping communication with computation could improve overall performance.

Second, there is potential to expand co-design exploration. Current domain decomposition in RES utilizes a uniform Cartesian grid, distributing equal-sized subdomains across CPUs, which does not account for hardware-specific characteristics like socket configurations, node boundaries, or the network topology. Hence, leveraging this type information for each system could lead to more efficient domain distribution and reduced communication overheads. To that end, further analysis of the communication patterns is planned to identify the type and frequency of data exchange between processes, which may reveal opportunities to reduce the volume of data transferred or the number of communication calls. Additionally, RES will explore the overlap of communication and computation phases and vectorizing computations to improve performance at a lower level.

3.2 Urban Air Project (UAP)

The purpose of the UAP pilot is to calculate airflow within cities. The most time and resource-demanding parts of this process are the different CFD workflows used during simulation. UAP uses three different codes for CFD: OpenFOAM [7] [8], Xyst [9] and RedSIM [10].

In D3.1 the benchmarks focused on the scalability of OpenFOAM and Xyst. All UAP codes showed high scalability, with OpenFOAM scaling up to 128 nodes in Discoverer and Xyst scaling up to 512 nodes in LUMI and MeluXina. Additionally, initial results were presented for RedSIM for both CPU and GPU execution, using up to 16 nodes for the CPU and 8 nodes for the GPU implementation, with good scalability.

This deliverable focuses mainly on: i) performance analysis and improvements for all UAP codes, and ii) extreme-scaling for OpenFOAM and Xyst.

3.2.1 Pilot progress and updates

UAP-FOAM

Table 8. UAP-FOAM benchmark workflow update

simulation part	execution	previous	new
data import	serial	once	once
mesh conversion	serial	once	once
decompose	serial	each #nodes	twice
renumberMesh	parallel	each #nodes	once
potentialFoam	parallel	each #nodes	once
simpleFoam [11]	parallel	each #nodes	once
changeDictionary	parallel	each #nodes	once

pimpleFoam [12]	parallel	each #nodes	each #nodes

To achieve higher scaling, focus was set on LUMI, for which special rules had to be followed resulting in updates to the benchmark workflow. Since LUMI does not support multiple parallel executions with one submit, the benchmarking process has been altered as such: most of the simulation workflow is done once on a single node, and only the final, most relevant part is run on multiple nodes. These changes are described in detail on Table 8.

To assess the limits of the UAP-FOAM code, parallel performance was measured by inserting new, more detailed timers in the OpenFOAM code to assess the average, minimum and maximum time for MPI Reductions, requests and wait operations. Figure 7 shows that all communication times decrease until about 32-64 nodes, after which point they stay relatively stable. Consequently, after that point communication does not scale, leading to communication-bound executions.

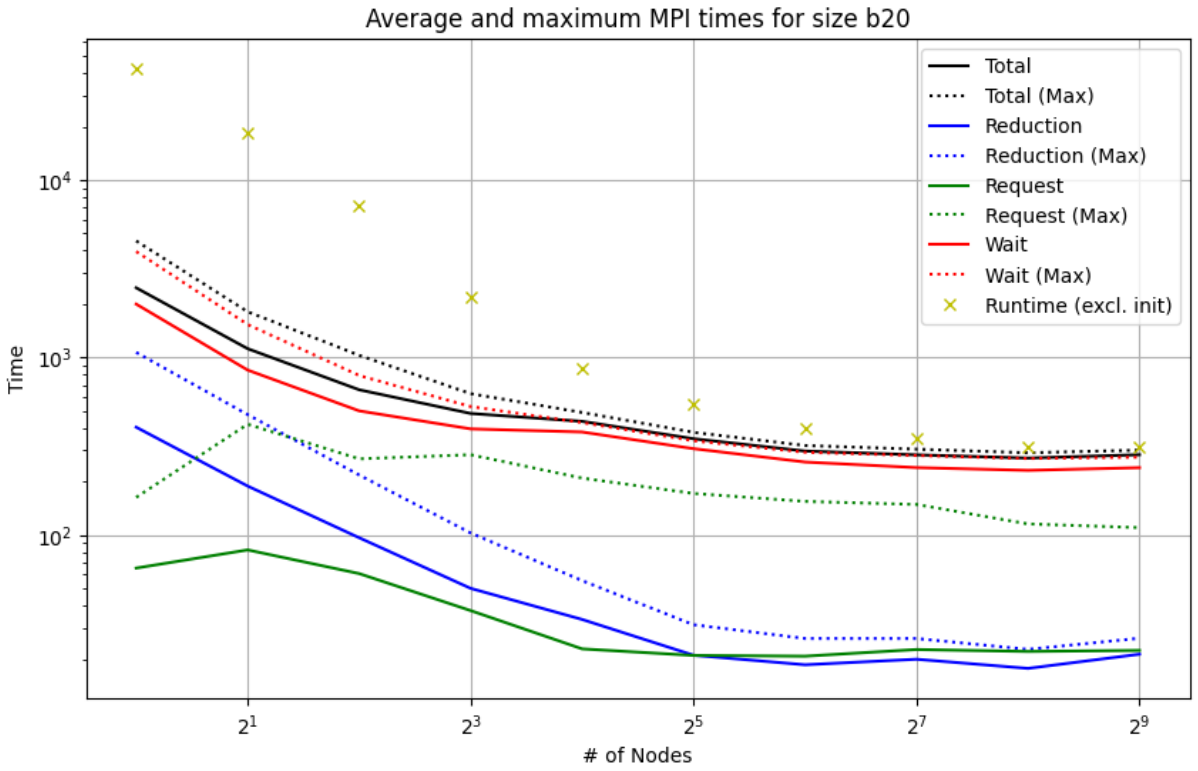


Figure 7. Various UAP-FOAM MPI call times averages and maximums. Call types with negligible or zero runtimes are ignored.

To delve deeper, mesh decomposition properties are shown in Figure 8. While communication times do not shorten, the number of cells and neighbouring cells are decreasing in number. Consequently, balancing domain decomposition is a valid approach for optimization, as currently domain decomposition focuses on distributing

cells evenly, contrary to process-to-process boundary cells. The differences in the number of neighbouring processes also suggests a possible opportunity for optimization.

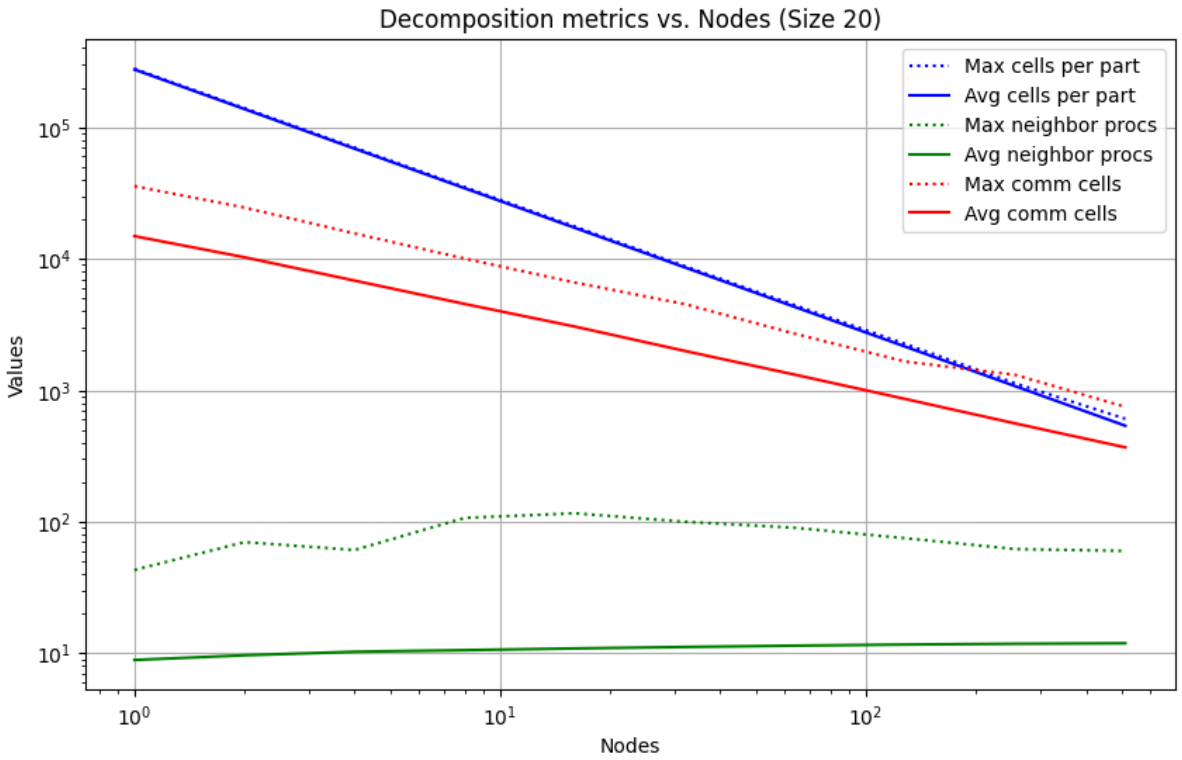


Figure 8. UAP-FOAM mesh decomposition metric averages and maximums.

UAP-RedSIM

Since RedSIM was at an earlier development phase during D3.1, its parallelization and performance optimizations have advanced rapidly, and are therefore explained in detail here. RedSIM is an iterative CFD solver, currently using first and second order explicit time-stepping methods as an integrator. Every simulation done in RedSIM can be broken down into two major phases:

- An initialization phase, where the polyhedral CFD mesh is partitioned across nodes, which includes setting up ghost cells and boundary conditions.
- A runtime phase, where RedSIM computes the derivative for each cell and does a step with the Explicit Euler method with a certain timestep.

RedSIM supports both CPU and GPU nodes for cluster computation, and is based on a hybrid MPI implementation: it assigns an MPI process per CPU socket, and manually handles multithreading via Linux APIs. This is especially important for the CUDA GPU

version of the code, since with GPUs it is impossible to go down to a thread level granularity with MPI.

While the initialization phase itself is important, the runtime phase is the most critical. RedSIM uses the Explicit Euler method with global time stepping to simulate unsteady flows. As a result, each iteration requires synchronisation of adjacent mesh cells between MPI processes, making mesh partitioning a critical factor for performance, since inefficient partitioning can lead to excessive communication between CPU nodes. Initial experiments used METIS and PARMETIS for partitioning; however, METIS is limited by its single-threaded implementation, and PARMETIS proved too slow in hybrid configurations. Based on these limitations, Zoltan's GRAPH [13] partitioning mode was selected as the preferred method for workload distribution. The partitioning is performed once during initialisation and remains fixed throughout the simulation. Additionally, the MPI_GRAPH communicator topology is used to inform the MPI runtime about major data exchange patterns between nodes.

Another problem due to the use of global time stepping is that each simulation step requires collective communication across all MPI processes at the end of an iteration. This is an inherent consequence of simulating unsteady flows and cannot be avoided within the current method. However, ongoing work explores a heuristic approach to estimate the global time step (τ) without requiring full communication between nodes, potentially reducing the associated overhead.

An example of the breakdown of MPI calls in a single iteration (without any exports / file writes in the iteration) is given below. This applies to the GPU version of RedSIM as well, with a few key differences mentioned later.

- *BEGIN SYNCHRONIZATION OF ADJACENT CELLS BETWEEN PARTITIONS.*
  - ***MPI_Isendrecv** called for each adjacent part*
- *END SYNCHRONIZATION OF ADJACENT CELLS BETWEEN PARTITIONS.*
  - ***MPI_WaitAll** called once.*
- *LOCAL TIME-STEPPING TAU MINIMUM REDUCTION*
  - ***MPI_Allreduce with MPI_MIN** called once.*
- *(OPTIONAL IF COMPUTING RESIDUAL) RESIDUAL COMPUTATION*
  - ***MPI_Allreduce with MPI_SUM** called once.*

RedSIM has been designed from the outset to support both GPU and CPU architectures and therefore does not require any porting. All CUDA kernels used for flux computations have been manually written and fine-tuned, with PTX assembly inspected and optimised at the instruction level. Early performance analysis was conducted using NVIDIA's NSight Compute toolkit on Windows, which helped identify initial hotspots. In multi-GPU configurations, the main performance challenge was heterogeneous communication latency, i.e., that communication between GPUs on the

same node is significantly faster than inter-node communication between CPUs. To account for this, RedSIM employs the MPI_GRAPH communicator topology in combination with OpenMPI's CUDA-aware MPI support. Future work is moving towards manual NVLink-based communication and workload management, in order to further improve performance for the GPU implementation.

UAP-Xyst

Work on Xyst continued after D3.1 on developing, testing, validating, verifying, documenting, and benchmarking solvers specialized to different types of flow problems. A high-level overview of the current solvers used in Xyst can be found on https://xyst.cc/inciter_main.html. The two latest solvers under verification and validation are ChoCG and LohCG, targeting constant-density (incompressible) flow.

3.2.2 Performance analysis

UAP-FOAM – System configuration

All executions of UAP-FOAM were conducted on LUMI. Table 9 details the programming and runtime environments used for all the runs.

Table 9. Programming & runtime environment for UAP-FOAM benchmarks

OpenFOAM	LUMI
Compiler	gcc/13.2
Parallel framework	cray-mpich/8.1.29
Libraries	
Boost	1.83.0
SCOTCH	7.0.4

UAP-FOAM – Benchmarking configuration

While UAP-FOAM model development is in the direction of supporting temperature and buoyancy, new mesh models are introduced to assess model accuracy, to improve adaptation to urban geometry and improve mesh generation automation. The meshes are generated using OpenFOAM's snappyHexMesh utility. Although the same geometry of Győr city is used as for previous benchmark models, resolution, cell sizes, and cell counts are different. While a very finely detailed class of 49 meshes were generated, only four are currently used in the benchmark. In the generation process, physical sizes and scales remain unchanged, as do refinement levels. The various cell resolutions are acquired by changing the base - coarsest - cell size, thus having different cell sizes at all refinement levels. Table 10 shows the new mesh properties.

Table 10. Details of UAP-FOAM benchmark meshes

#	name	type	cell count
1	b20	polyhedra	35.2M
2	b40	polyhedra	10.6M
3	b80	polyhedra	2.83M
4	b160	polyhedra	896k

UAP-FOAM – Results & analysis

The results depicted in Figure 9 show the speedup progression from 1 to 512 CPU nodes (128-65536 cores). Only pimpleFoam, the simulation part, is benchmarked. The time between the first and last iteration is measured, skipping initialization which is minimal. As expected, the finer grids scale better, since they entail more computation. The speedup for the highest mesh size increases in a superlinear fashion up to 128 nodes, above which point it reaches a plateau.

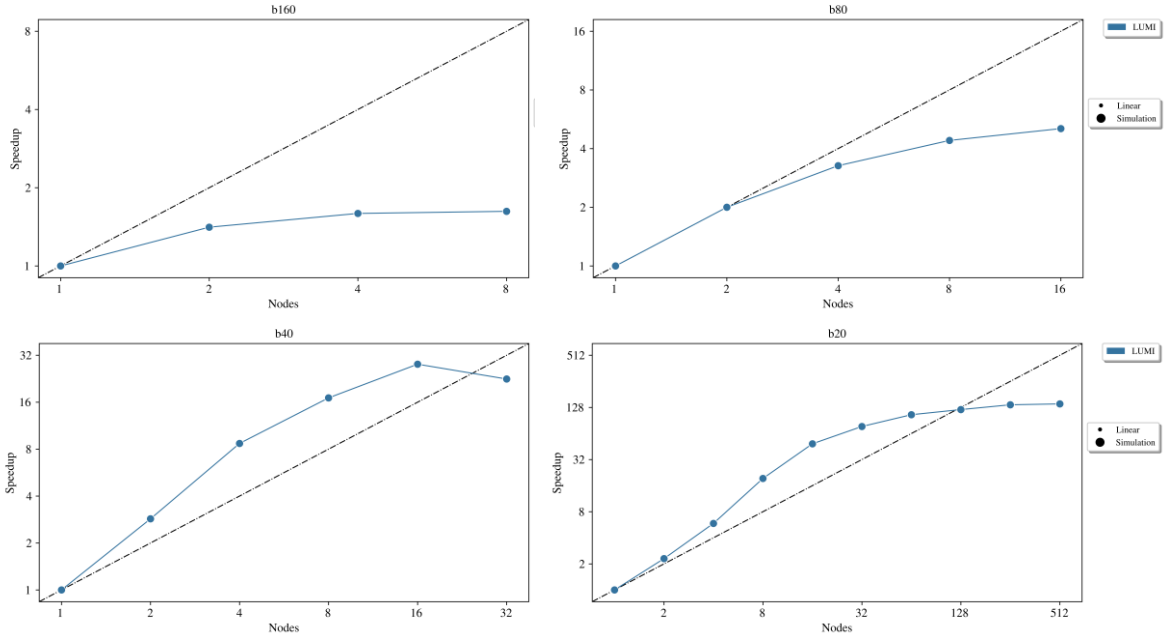


Figure 9. Per node speedup values for pimpleFoam on various number of nodes on LUMI. Multiple mesh sizes are shown from coarsest (b160, top left) to finest (b20, bottom right).

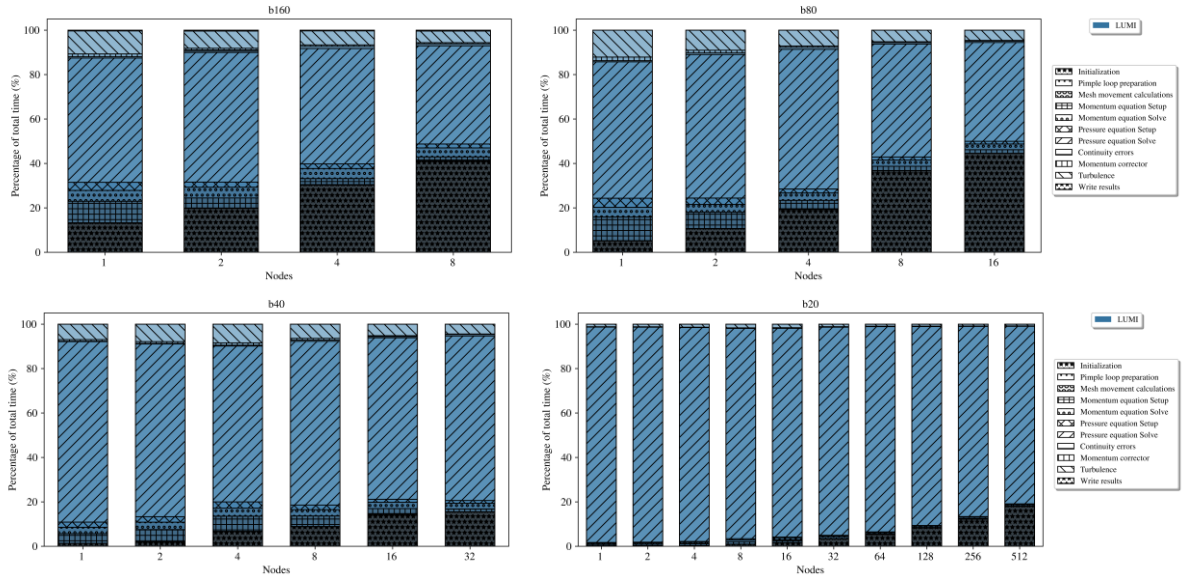


Figure 10. Per node execution breakdown for pimpleFoam on various number of nodes on LUMI. Multiple mesh sizes are shown from coarsest (b160, top left) to finest (b20, bottom right)

To gain more insight on the execution as a whole, regional timestamping and intrinsic profiling using std::chrono::high_resolution_clock was used in the code. As Figure 10shows, initialization and solving the pressure equation are the most dominant factors in runtime. From these two, since only the pressure equation solver scales with simulated time (the initialization is run only once), it is reasonable to focus on that to improve performance.

UAP-RedSIM – System configuration

UAP-RedSIM has been executed on the CPU and GPU partitions of Karolina. Table 11 provides the details of the programming and runtime environments used for all runs.

Table 11. Programming & runtime environment for UAP-RedSIM benchmarks

RedSIM	Karolina-CPU	Karolina-GPU
Compiler	gcc 12.2	cuda 12.3
Parallel framework	openmpi 5.0.5	openmpi 5.0.5

UAP-RedSIM – Benchmarking configuration

Benchmarks for RedSIM were performed in the context of Urban Air Pollution for the city of Gyor, with 2, 10 and 30 million cell mesh sizes, both for CPU and GPU as described in

Table 12. All simulations were first order in time and first order in space. The CPU RedSIM code was run using ParMETIS, while the GPU with Zoltan GRAPH.

Table 12. Details of UAP-RedSIM benchmark meshes

City	Mesh size	Iterations
Gyor	2M	10000
Gyor	10M	5000
Gyor	30M	1000

UAP-RedSIM – Results & analysis

Figure 11 and Figure 12 depict the speedup observed for the CPU and GPU implementations of UAP-RedSIM respectively. RedSIM-CPU scales well up to 64 nodes (8192 cores) for 30M cells, and 32 nodes for smaller mesh sizes. RedSIM-GPU also showcases remarkable performance, scaling up to 32 GPUs for the 30M mesh size and 16 GPUs for 10M mesh size, demonstrating scalability beyond intra-node, as Karolina has 8 GPUs per node. Inconsistent behaviour was registered for 16 GPUs at 2M mesh size. In comparison with D3.1, the CPU parallel version was reworked from scratch. The original version worked in a master-slave model, running one MPI process on each of the slave nodes, and OpenMP threads for intra-node parallelization. The new implementation benchmarked for this deliverable RedSIM uses a completely democratic code for the CPU implementation, using no master node and running two processes per node (one per socket) and OpenMP threads for parallelizing within the sockets. This fundamentally new approach resulted in completely different scaling behaviour. Alas, the code still produces a reasonably good performance up to 64 nodes.

In comparison with D3.1, the GPU version also got completely reworked. The original version was based solely on CUDA, and while it supports intra-node parallelization within its API, maximum number of GPU's was limited to 8, provided by Karolina's GPU node. The new hybrid MPI-CUDA based implementation supports multi-node execution, thus allowing runs up to more than 8 GPUs. All resulted GPU performance was excellent within one node. The 10M mesh scales up to 16, and the 30M mesh up to 32 nodes almost linearly in this section.

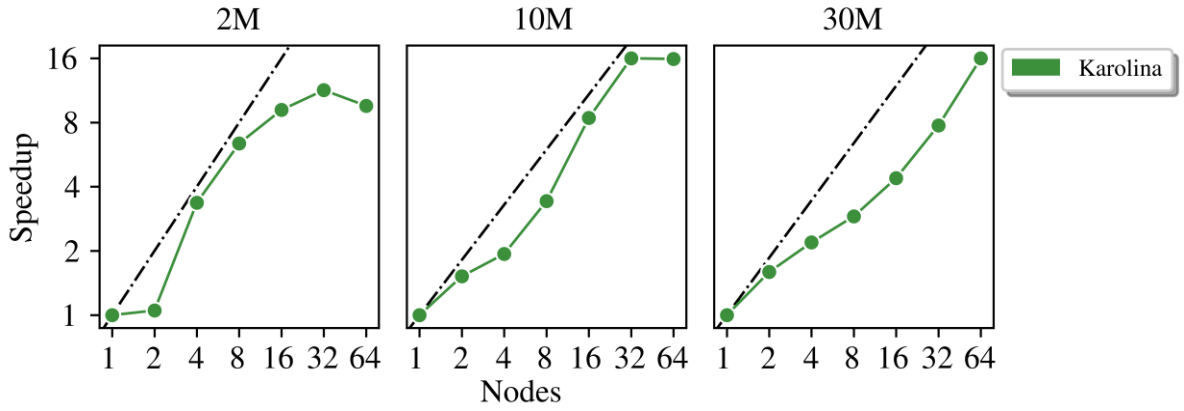


Figure 11. Per node speedup for the MPI parallelized CPU version of RedSIM

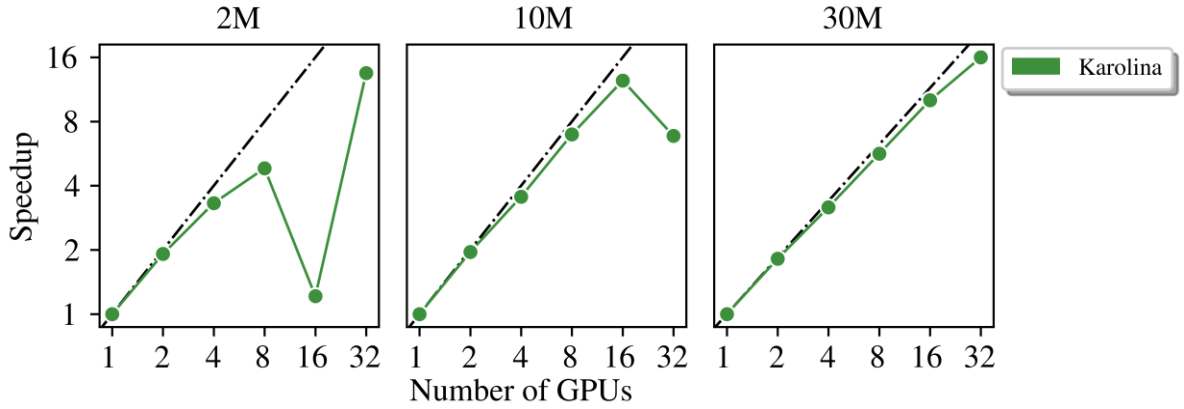


Figure 12. Speedup for the MPI-CUDA parallelized GPU version of RedSIM

UAP-Xyst – System configuration

Performance tests for Xyst have been carried out on LUMI using the configuration described in Table 13.

Table 13. Programming & runtime environment for UAP-Xyst benchmarks

Xyst	LUMI
Compiler	gcc/12.3
Parallel framework	cray-mpich/8.1.29
	Charm++ v7.0.0.rc2
Libraries	
mpich-ofi	12.3
netCDF	4.9.0

UAP-Xyst – Benchmarking configuration

All UAP-Xyst benchmarks were done with a 133M points, 749M cell count tetrahedral mesh. For benchmark model for the RieCG and ZalCG compressible solvers, the Taylor-Green vortex problem was used. With the LohCG incompressible solver the lid-driven cavity flow field was computed.

UAP-Xyst – Results & analysis

The benchmarks include running three Xyst solvers above 65K CPU cores for the first time, as LUMI "**hero**" runs. These computed verification problems of academic interest, and were meant to stress-test the entire Xyst I/O, startup, time stepping, and asynchronous message passing infrastructure up to computational meshes with approximately 800-million elements, corresponding to over 100-million nodes (solver degrees of freedom, DOF). Strong scaling up to almost 200-thousand CPU cores have been established for two of the solvers for the first time.

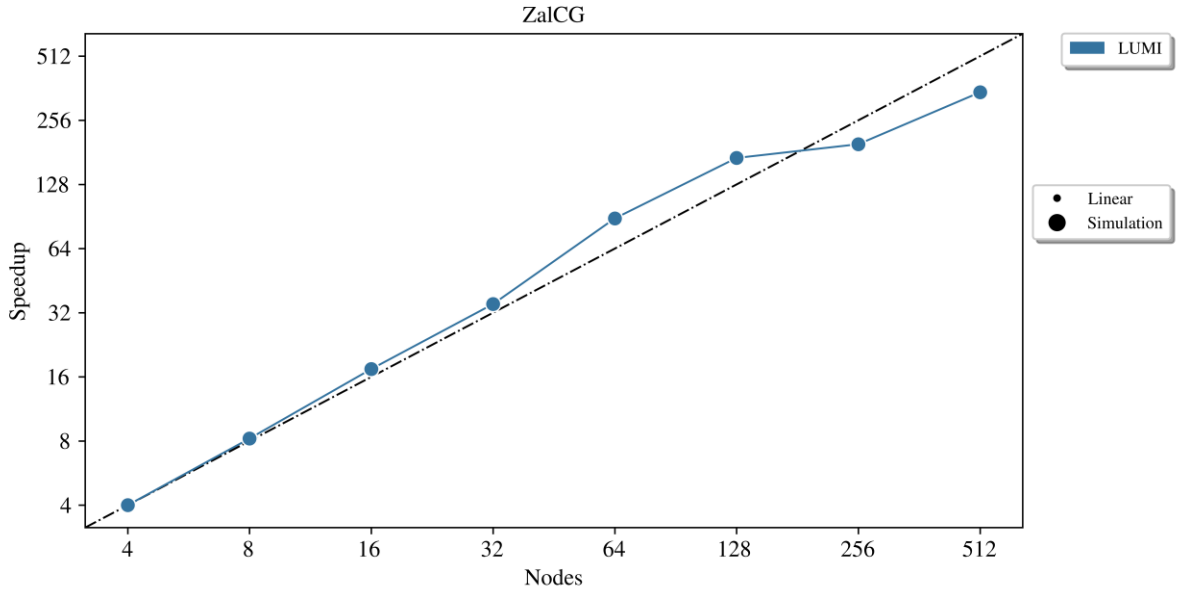


Figure 13. UAP-Xyst per node speedup for the ZalCG solver on LUMI

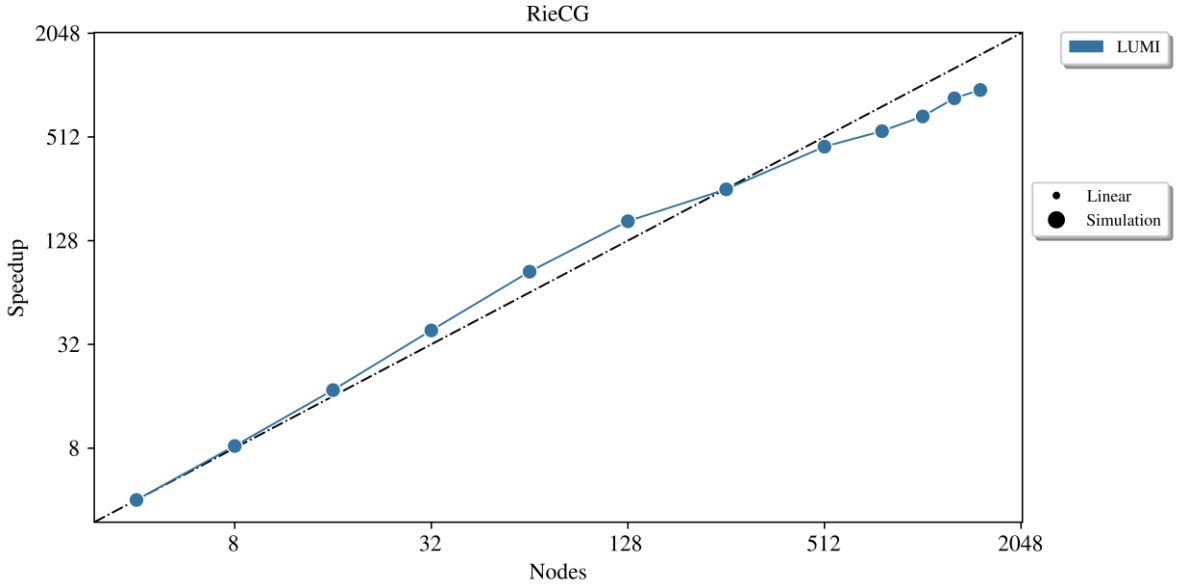


Figure 14. UAP-Xyst per node speedup for the RieCG solver on LUMI.

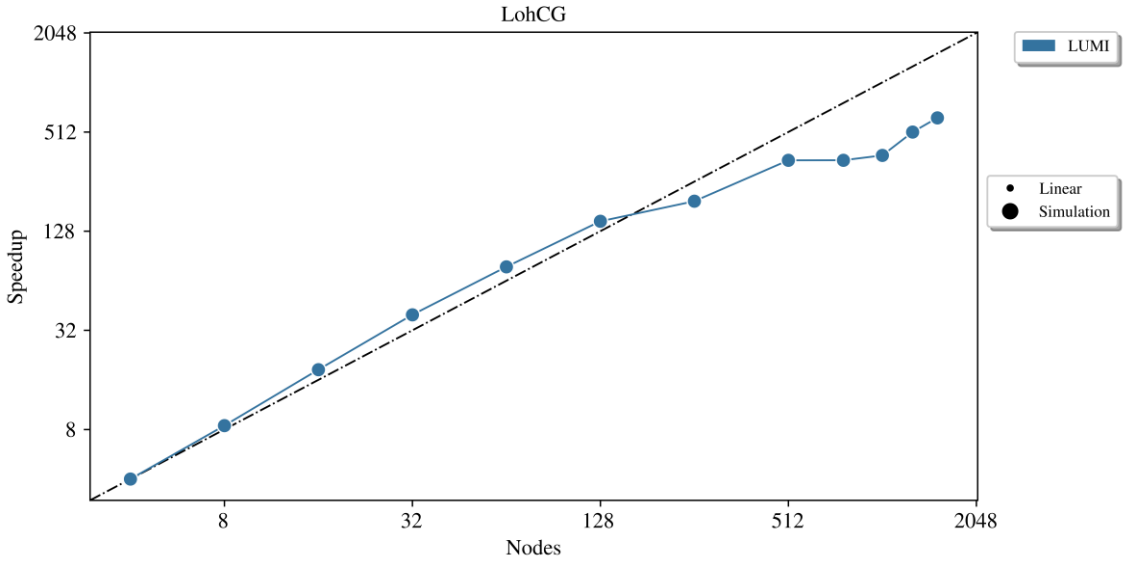


Figure 15. UAP-Xyst per node speedup for the LohCG solver on LUMI.

Figure 13, Figure 14 and Figure 15 show scaling results for three different solvers: ZalCG, RieCG and LohCG, respectively. For all solvers, a minimum of 4 nodes were benchmarked, achieving a speedup of 4x. As it is evident in the figures, Xyst scales for all solvers in a superlinear fashion up to 128 nodes and becomes slightly sublinear beyond that point.

In general, it is evident that Xyst can cope with problems of O(10^8) DOF using compute resources of O(10^5) CPUs across thousands of networked compute nodes. Strong scaling does not yet reach the plateau of diminishing returns for any of the solvers tested even at the largest core counts of almost 200K CPUs for RieCG and LohCG. Additionally, the largest runs corresponded to about a thousand DOF/core

which means runtime could potentially still be decreased using even more resources, if available.

As a result, the absolute performance for the LohCG incompressible-flow solver is below 4 milliseconds per time step with a O(10^8) DOF problem without reaching a scalability bottleneck. More information about these results can be found at [40].

UAP - Summary and next steps

The latest UAP deliverable shows that all three CFD codes—OpenFOAM (UAP-FOAM), RedSIM and Xyst—scale well up to hundreds of nodes, with OpenFOAM achieving superlinear speedups to 128 nodes on LUMI before plateauing, RedSIM's CPU and GPU versions scaling to 64 CPU nodes and 32 GPUs on Karolina, and Xyst's solvers demonstrating superlinear scaling to almost 200 000 cores on LUMI with no bottleneck yet. Detailed MPI timing in UAP-FOAM revealed that communication costs level off beyond 32–64 nodes, suggesting that adopting boundary-aware or adaptive mesh decompositions could reduce inter-process traffic. RedSIM's shift to Zoltan GRAPH partitioning and an MPI_GRAPH topology has unlocked near-linear scaling, and Xyst's hero runs on a 749 M-cell mesh confirm its ability to handle O(10^8) DOF at extreme scale.

3.3 Urban Building (UB)

The Urban Building (UB) (aka Ktirio Urban Building [14]) pilot's purpose is to simulate the energy behaviour of buildings at scales; ranging from building scale to entire cities and beyond. During execution, the simulation estimates each building's thermal comfort, energy consumption and air quality, and the goal is to generate these predictions over periods ranging from one month to a full year, reflecting a realistic environment. This involves incorporating factors such as weather conditions, occupancy patterns, and surrounding vegetation.

The pilot considers different degrees of accuracy, referred to as the Level of Detail (LOD). In the UB context, these representations of buildings are classified as:

- LOD-0: Buildings are represented as oriented bounding boxes.
- LOD-1: Buildings are represented as multi-polygonal extrusions, optionally including a catalogue of roof shapes.
- LOD-2: Buildings are detailed from an Industry Foundation Classes [15] (IFC) description encompassing many intricate details.

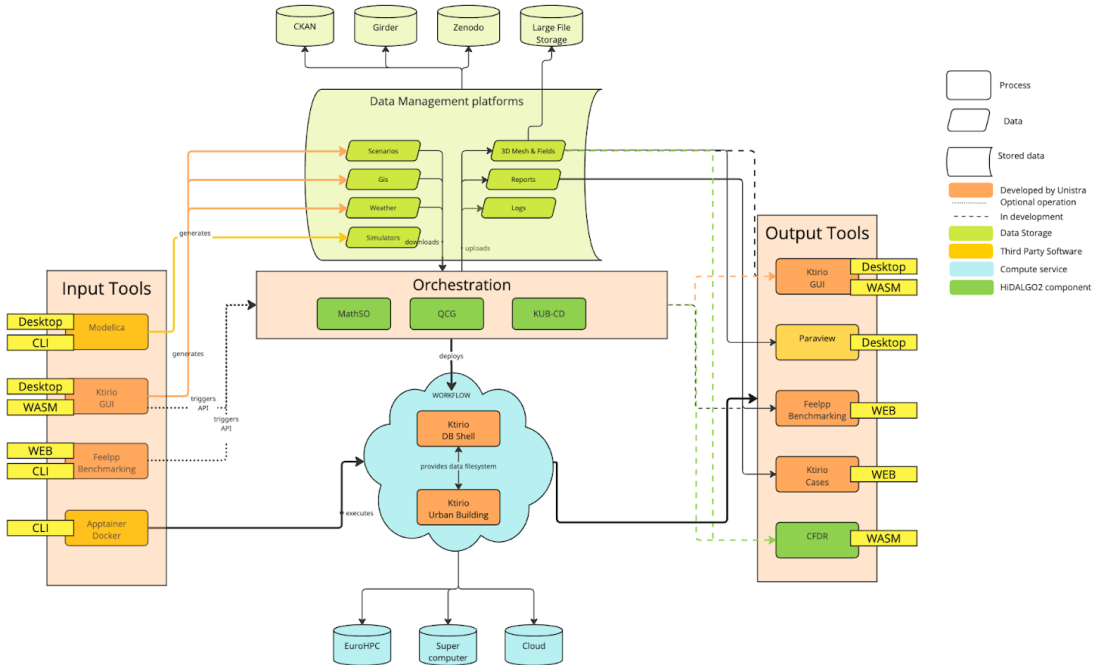


Figure 16. UB pilot code pipeline

The UB simulation pipeline can be separated into the following steps as seen on Figure 16:

- **Input data generation (using Ktirio-GUI)**: The city energy simulation requires the geometric description of a geographic area containing buildings. To do so, the Ktirio Graphical User Interface takes in charge the generation of GIS (JSON) and mesh files, in LOD-0 and LOD-1 representations. This is done by using open databases on the web, such as OpenStreetMap [16], which allow including more realistic factors such as the terrain topography and the surrounding vegetation. The Ktirio-GUI is built using the QT framework and C++, and supports multi-threading. It not only handles physical components, but also requests and processes weather information for the selected zones, and creates different occupancy scenarios depending on the different building types.
- **Building Model definition**: Using the Modelica [17] language, physical systems are modelled. The models are then translated to C++ applications using the Functional Mock-up interface [18] (FMI).
- **City Energy Simulator**: A C++ library designed to simulate a city energy model, based on the Feel++ [19] framework. The simulation is parameterized by GIS data, different LOD meshes, building models, occupancy scenarios, and weather conditions. This application is parallelized on CPU nodes based on a distributed approach using MPI.

To guarantee and automate the simulation execution on different HPC systems, the UB pilot takes advantage of containerization, using Docker [20] and Apptainer [21]. This way, the UB programming environment is reusable at any time and is independent of the underlying machine, ensuring easy deployment and reproducible results.

In the previous deliverable (D3.1), a scalability study of Ktirio-UB was performed by deploying the pilot's pipeline using 1-32 nodes (with 128 processes per node), which showed that the simulation component of the workflow scaled almost linearly but the total execution time did not scale and performance degraded when increasing compute nodes. This degradation was caused by a bottleneck concerning the post-processing stage of the pipeline. The source of this problem was found to be multiple files being written in parallel on the shared file system. Specifically, it was identified that opening and closing output files was occupying most of the pipeline's total execution time and did not scale.

3.3.1 Pilot progress and updates

UB – Code and pipeline changes

Multiple features were implemented since D3.1 in the workflow in order to improve the accuracy of the city energy simulations. These will be presented at length in deliverable D5.7, since they are not optimizations targeting performance or scalability. A brief description of these features is the following:

- The computation of shading masks, which is important for accurately modelling the impact of solar radiation on building surfaces.
- The addition of vegetation objects in the neighbourhood's or city's geometry, which has a significant impact on the energy consumption of buildings along with solar masking.
- The implementation of a heating control loop for each building supporting boilers and heat pumps. Previously, UB supported only ideal heat systems.
- The addition of a post-processing step that aggregates and exports only the simulation outputs of interest for each experiment. This feature is very vital in reducing the computational cost of I/O operations for simulations where the complete and detailed simulation results dataset is not needed.
- The introduction of more precise timers in order to measure the performance of more fine-grained sections of code.

UB – Profiling, bottlenecks and code/algorithm improvements after D3.1

After D3.1, in order to analyse and optimize the performance of the UB code, profiling was conducted. The initial attempt was with Eztrace from NumPEx, the French

exascale initiative, which was chosen for its ease of integration. Eztrace uses LD_PRELOAD, meaning it requires no recompilation, which was essential given the complexity of the UB build pipeline. However, the profiling attempt was unsuccessful due to several bugs resulting in traces not being visualized correctly. These issues have been reported to the Eztrace development team, and are still under investigation.

Since UB's build pipeline complexity is very high, alternative tools like Score-P which require re-compilation could not be easily integrated to the existing setup. To address this, the UB pilot has introduced a new packaging system known as Spack into their workflow which will ultimately enable Score-P profiling support. Although Score-P is not yet available for integration at the time of D3.2, its inclusion will provide a more robust profiling option for future iterations.

Even without profiling, the primary bottleneck for the UB pipeline was already identified in D3.1; post-processing did not scale due to the high cost of writing HDF5 files across a large number of cores. To address this, the workflow was modified to generate simulation reports only for relevant information, defined during simulation execution, which reduced the actual volume of the output size. reducing communication overhead and enabling more targeted performance analysis. Then, the pre-processing stage was restructured for distributed partitioning, dividing the input dataset among MPI ranks to minimize memory constraints and improve scalability.

3.3.2 Performance analysis

UB – System configuration

As mentioned previously, UB uses containers to ensure that the programming environment is reusable and independent of the underlying machine. UB executions were performed on 4 different EuroHPC JU systems: Discoverer, Vega, Karolina and Meluxina for which the programming and runtime environments are detailed on Table 14. The UB pilot performance measurements are based on the feelpp.benchmarking framework, which eases the benchmarking process of any application on HPC systems through the generation of comprehensive reports.

Table 14. Programming & runtime environment for UB benchmarks

	Discoverer	Vega	Karolina	Meluxina

Compiler (container)	Clang 14			
Parallel framework	OpenMPI 4.1.6	OpenMPI 4.1.5	OpenMPI 4.1.4	OpenMPI 5.0.3
Libraries				
Apptainer	SingularityPRO version 3.7-4.el8	SingularityPRO version 4.1.6-1.el8	1.3.6	1.3.6
Python	3.9.7	3.10.8	3.10.4	3.11.10

UB – Benchmarking configurations

Previously, the scalability of the UB pilot's workflow was evaluated on Discoverer, Karolina and MeluXina, for two different area sizes of a square centred in Strasbourg. The datasets contained approximately 6000 and 17000 buildings for square side sizes of 2km and 4km respectively. As multiple features were included since deliverable D3.1, resulting in major modelling changes, new benchmarking scenarios are considered for D3.2 in order to evaluate the new workflow's strong and weak scaling. Table 15 describes the different benchmark scenarios and their parameters. All benchmarks are performed using a LOD0 mesh and a 1-day timespan during winter, corresponding to January 1st, 2024.

Table 15. New benchmarking scenarios for UB

	Location	Heating systems	Quadrature Order	LOD	Period	Radius	# nodes
S1	Paris	Ideal	3	0	1 day, winter	1 – 6 km	2 – 50
S2	Paris-Berlin	Ideal	3	0	1 day, winter	5 km	2 – 50
S3	Paris	Ideal	0 – 5	0	1 day, winter	3 km	2 – 10

More specifically, three different scenarios were used:

- **S1**: The first scenario focuses on reviewing the pipeline's spatial and computational scaling. It does so by varying the radius of the circle centred on Paris that defines the mesh, from 1km to 6km with a 1km step, as well as the number of compute nodes used for the simulation, using 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40 and 50 nodes, with 128 tasks per node. Some large cases required more memory than the memory allocated, therefore, the number of tasks per node was reduced.
- **S2**: The second scenario's objective is to study the impact of the building density of a city in energy simulations, specifically concerning solar shading. For this, a

3km radius circle centred on Paris and on Berlin will be considered. The computational resources used will vary from 2 nodes to 50, as in the previous scenario.

- **S3**: The third scenario assesses the impact of the accuracy of the solar mask evaluation. The quadrature order parameter controls the number of rays used in the ray tracing algorithm: the greater the order of quadrature, the greater the number of rays. The performance of the solar masks and building simulation components will be analysed depending on the quadrature order, for the 3km mesh of Paris, using 2 - 10 nodes and considering a 1-day period.

Some relevant statistics for the input meshes for the different radius values are shown in Table 16 and Table 17.

Table 16. Characteristics for the Paris dataset

Paris	Nb buildings	Nb vertices (LOD0)	Nb triangles (LOD0)
1km	4 770	185 341	323 140
2km	17 455	734 992	1 287 744
3km	36 339	192 5476	3 394 036
4km	65 390	3 940 001	6 968 494
5km	102 506	5 856 068	10 337 432
6km	150 292	7 524 817	13 243 994

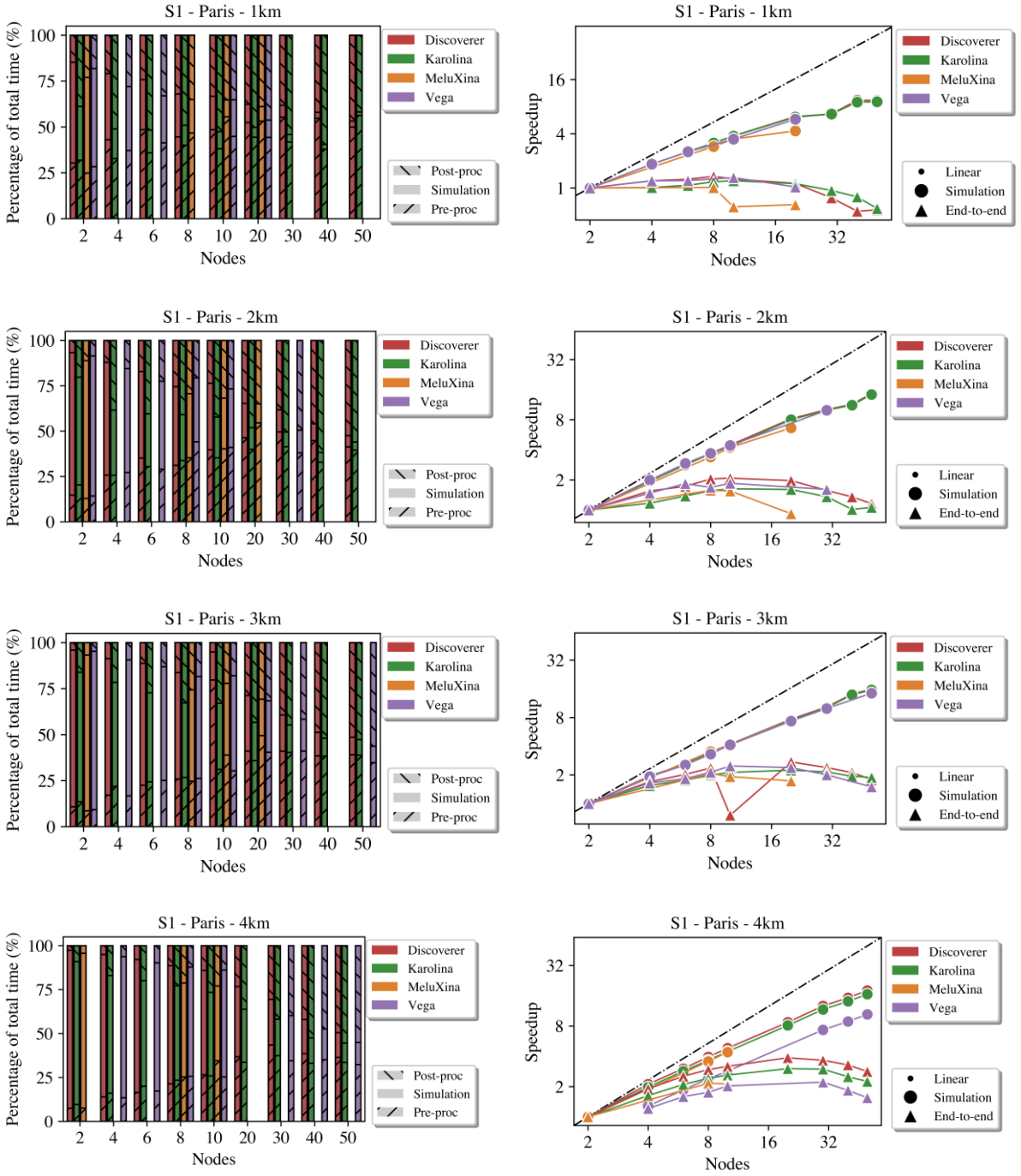
Table 17. Characteristics for the Berlin dataset

Berlin	Nb buildings	Nb vertices (LOD0)	Nb triangles (LOD0)
1km	3 842	128 465	223 400
2km	8 186	471 101	830 828
3km	17 505	1 103 919	1 949 136
4km	32 128	1 959 435	3 466 526
5km	51 007	3 165 351	5 604 338
6km	77 046	4 382 797	7 735 780

UB – Results & analysis

In order to accurately compare the application's performance to D3.1, the solar masks, aggregated output report and visualization export components have been deactivated. Consequently, the pre-processing stage mainly consists of reading partitioned city meshes, and parsing files related to weather conditions and occupancy scenarios. The simulation stage only considers the building simulation component, and the post-processing stage represents only the export of output quantities in parallel using the HDF5 format.

Figure 17 shows the performance analysis for the S1 scenario for these adjustments, using 2-50 nodes and 128 tasks per node, for values of the input city mesh radius ranging from 1km to 6km. The figure depicts the performance breakdown for pre-processing, simulation and post-processing *(left)* and the speedup achieved *(right)* on each EuroHPC JU machine for the total execution (*end-to-end*) and the simulation component of the pipeline (*simulation*).



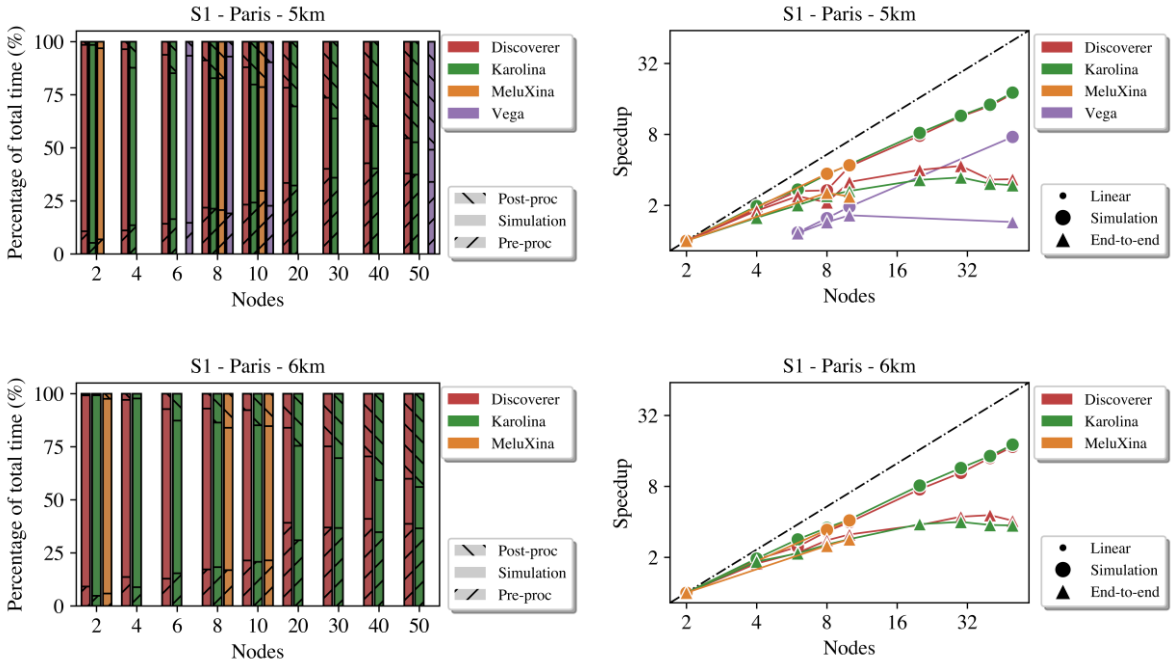


Figure 17. UB-Ktirio performance results for the Paris scenario, with only the D3.1 modelling components active for 1 - 50 nodes

First, it is notable that in general the speedup improves with the problem's size, e.g. the city radius. For all city radius values the simulation part of the workflow takes the most time on low numbers of nodes. However, as node count increases, the execution time of the simulation starts being dominated by the pre-processing and post-processing steps.

In contrast to D3.1, the simulation stage has a slightly lower speedup, which is attributed to more advanced building models being used, while the partitioning procedure does not yet account for this added complexity. More specifically, while buildings are assumed to have the same weight in the load balancing, the current urban building model has been improved with various characteristics and differences depending on the building type, like the number of floors.

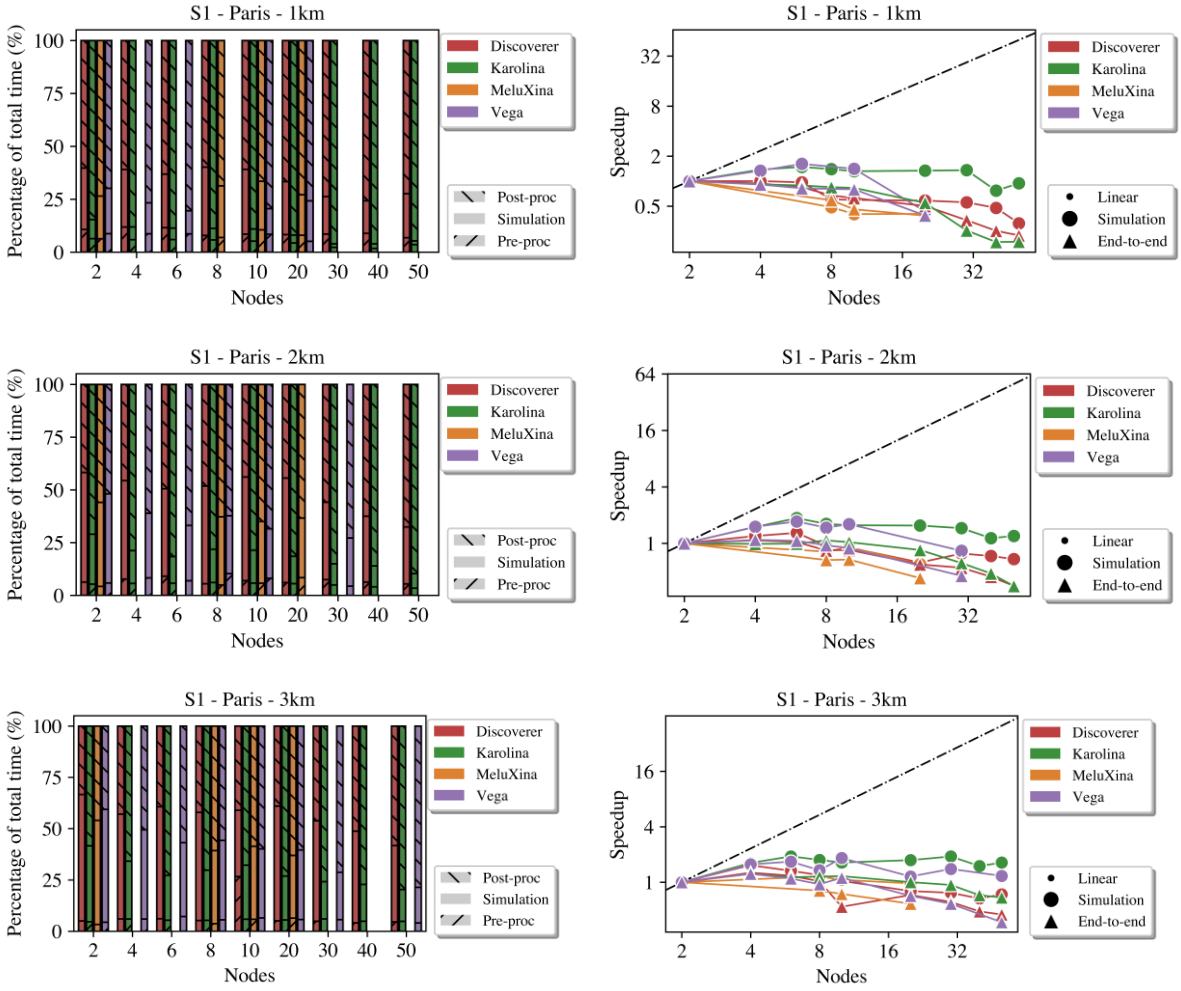
On the other hand, the end-to-end pipeline now shows some scaling with the computation resources, while in D3.1 it actually resulted in slowdowns. The most important factor for this is that computation time needed for exporting HDF5 outputs has significantly been reduced, leading to an important improvement in the total pipeline execution.

Next, to assess the performance of the updated code, all newly implemented features of UB were enabled, allowing the analysis of the solar masks component, as well as the aggregation report and visualization export.

Figure 18 presents the performance achieved for the S1 scenario, i.e., when using increasing mesh sizes. It is evident that enabling the new features leads to a significant

impact on the performance. Improved scalability is traded off in favour of incorporating more realistic components and thus obtaining more accurate results.

It can be seen that there exists a case (3km, 10 nodes) where the pre-processing stage takes longer than usual for Discoverer, which can only be explained by stability issues with Discoverer's file system. Additionally, some benchmarks on Vega were unsuccessful, either because of very long queue times or due to connection issues concerning SLURM. In general, executions on Karolina exhibited better performance and scaling compared to the other systems.



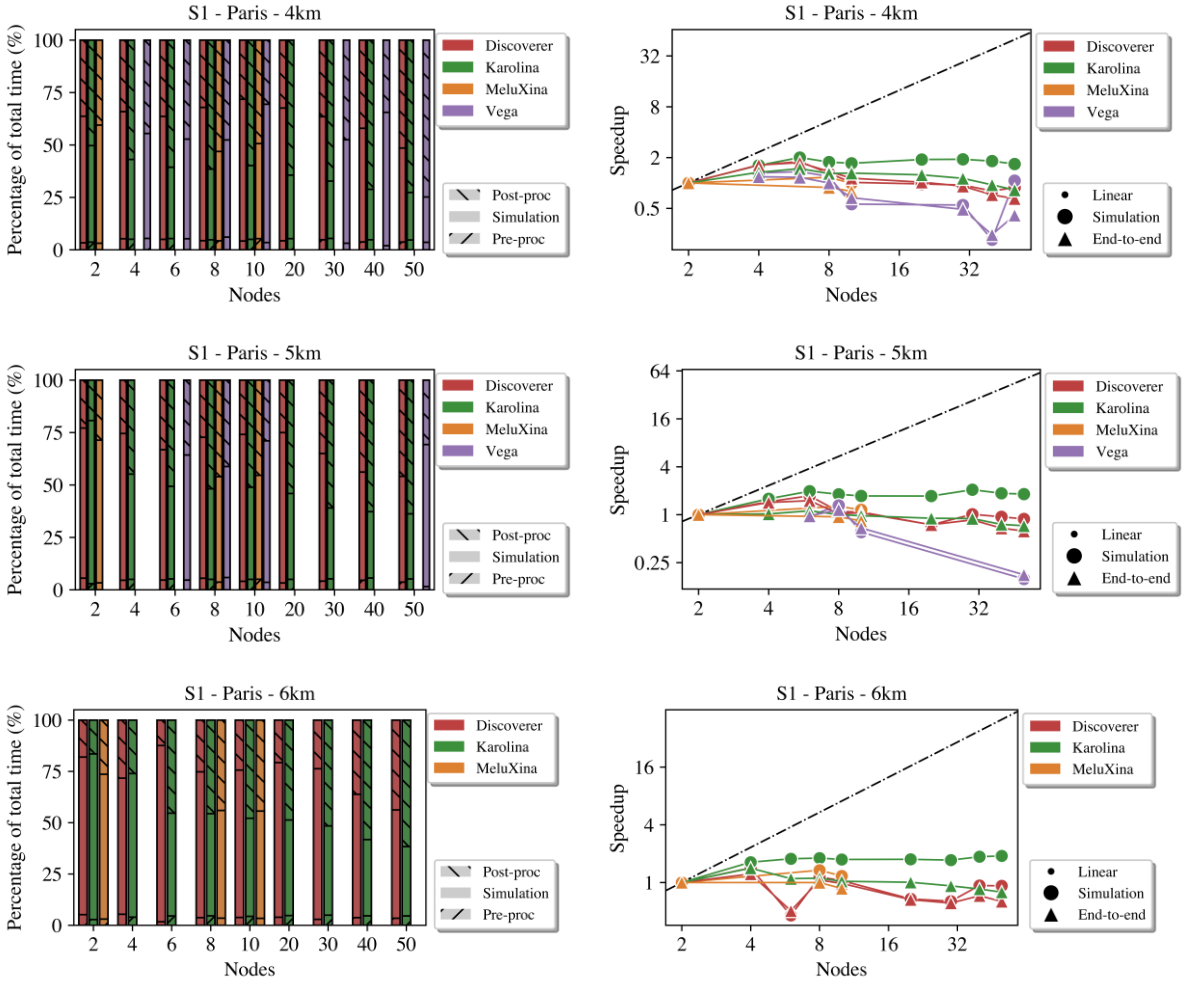


Figure 18. UB-Ktirio performance results for the Paris scenario, with all modelling components active for 1 - 50 nodes

The benchmarking has provided a detailed insight into the efficiency and scalability of UB's components. More specifically, both areas of success and others requiring further optimization have been identified:

- **Solar Mask Evaluation**: The evaluation of the solar mask shows significant scaling issues. When a large number of MPI tasks is employed, the computational time increases dramatically. This performance degradation was anticipated given that the current implementation, while robust, is not fully optimized for MPI parallelism. To address this, work has begun on incorporating spatial partitioning, which will localize solar mask intersections within designated processor groups. This approach is expected to reduce communication overhead and improve scalability.
- **Building Simulation**: The building simulation component exhibits near-perfect scaling. Task distribution across nodes is well-balanced, and the simulation

performs efficiently under parallel execution. This confirms that the underlying model and its MPI task management are functioning as intended.

- **Post-Processing Steps**:
  - **Export Outputs (HDF5)**: The export of simulation outputs in HDF5 format consumes minimal time, indicating that this process does not significantly impact the overall execution.
  - **Report Generation**: Report generation maintains good scaling behaviour even as computational resources increase.
  - **Export Visualization (Ensight)**: In contrast, the export of large visualization files in Ensight format shows a marked increase in execution time with the number of nodes used. This suggests that parallel access to shared file systems is a bottleneck. Alternative strategies, such as caching intermediate results or leveraging local disk storage, are being considered to alleviate this issue.

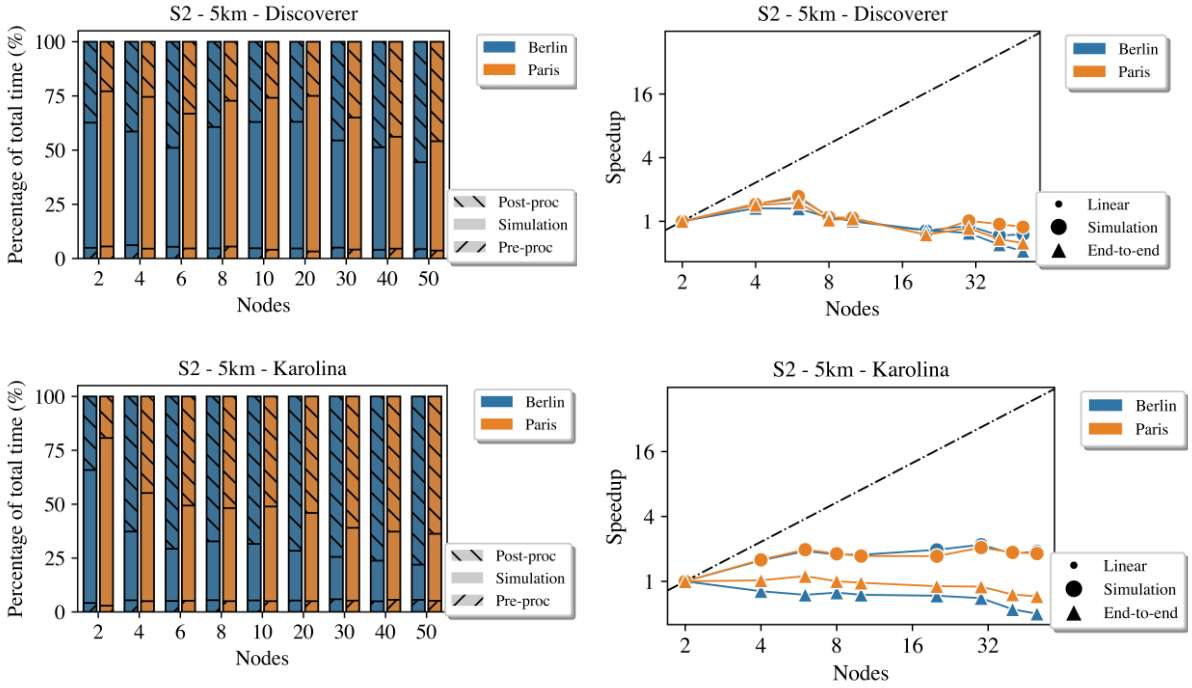


Figure 19. UB-Ktirio performance results for the Paris and Berlin-Paris scenarios, with all modelling components active for 1 - 50 nodes

Figure 19 depicts the results for the S2 scenario, i.e., for two considerably large European cities (Paris and Berlin) that have significantly different densities concerning building distribution. Even if the relative time taken by the simulation varies between cities (caused by the difference in the number of buildings), the speedup is almost identical between Paris and Berlin. This leads to the conclusion that the density of a city has no major influence on the application's scaling.

Neither case achieves linear scaling for the simulation stage, implying that UB pipeline's scaling might be independent of the input dataset. On the other hand, due to the higher number of buildings present in the Paris dataset, post-processing takes significantly more time than Berlin with respect to the other phases of the workflow. Finally, by leveraging pre-computed mesh partitions, the pre-processing stage seems to be also independent of the input dataset. The time percentage required to load input remains almost constant.

Finally, the third scenario (S3) assesses the impact of quadrature order on performance, taking into account the maximum computation time of all iterations, the total computation time and the relative performance of the solar masks component. All experiments were performed on the Discoverer and Karolina clusters, for 2-10 nodes, on the 3km - Paris mesh, for the quadrature order taking values from 0 to 5.

Figure 20 shows the maximum computation time over all iteration and the total execution time that the solar masks component took, running on 8 nodes of the Discoverer cluster. It can be deduced from the figure that the number of quadrature points has a significant impact on the computation time. Considering a quadrature order of 3 instead of 4 reduces the solar mask computation time by half. A more in-depth analysis should be done on the accuracy of solar shading coefficients to examine if a high order is necessary.

Finally, Figure 21 depicts the relative performance and speedup of the simulation component (solar masks and building energy simulation) for the given quadrature orders and for 2 to 10 nodes, on the 3km - Paris dataset. It is evident that the simulation component scales better for lower quadrature orders, notably when executing on a larger number of nodes. In order to profit from the accuracy of a higher number of quadrature points, mesh partitioning should be revisited to improve the scaling of the solar masks component, along with optimizing MPI communications.

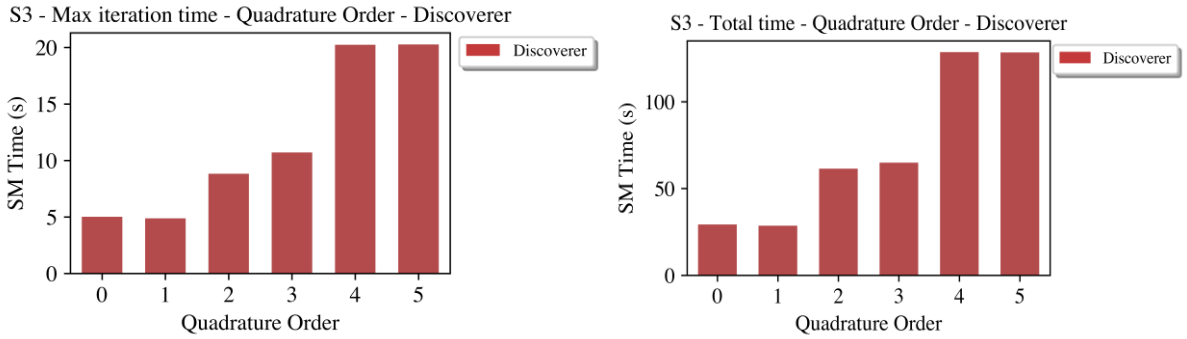


Figure 20. The number of quadrature points and their effect on execution time for the solar mask component

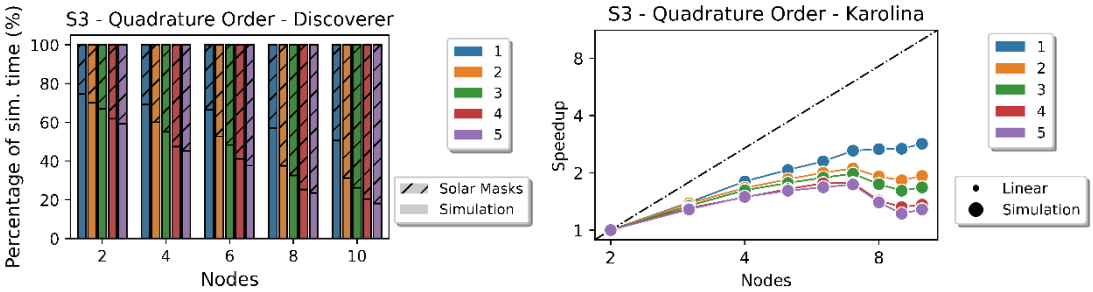


Figure 21. Relative performance and speedup of UB simulation by quadrature order

UB - Summary and next steps

Concluding, the UB pilot has redesigned its codebase, adding new more complex model components, and extended benchmarking across multiple EuroHPC JU systems. End-to-end scalability has improved considerably for the model workflow components used in D3.1, largely due to reduced output export time in the post-processing stage. On the other hand, the simulation speedup was slightly lower than in previous results, which was attributed to increased model complexity that is not yet fully accounted for in the load balancing strategy. Regarding new model components introduced after D3.1, the solar mask evaluation and visualization export stages emerged as new bottlenecks while the pre-processing performance was largely stable and independent of the input dataset size.

Moving forward, optimisation will focus on improving the MPI parallelism of the solar mask component, using spatial partitioning to reduce communication overhead. The Ensight-based visualization export will also be reviewed, with strategies such as caching intermediate data or using local storage to reduce I/O contention. Further analysis will evaluate the trade-offs between quadrature order and solar shading coefficient accuracy, aiming to achieve a balance between performance and model fidelity. These improvements will be integrated in the next development phase to support more scalable and robust large-scale UB simulations.

3.4 Wildfires (WF)

The Wildfires (WF) pilot focuses on simulating fire propagation and its interaction with the atmosphere across different spatial scales. Within the pilot, two primary use cases are considered, based on the resolution and the characteristics of the simulated scenarios: the landscape level and the settlement (urbanization) level.

- Landscape level: The objective at this level is to simulate wildfire progression and its coupled interactions with atmospheric dynamics. This includes modelling the release of energy, wind field disturbances, pyro-convective phenomena, and the emission and dispersion of smoke. The modelling approach combines the Weather Research and Forecasting model [22] (WRF) with SFIRE [23], a semi-

empirical fire behaviour module tailored for coupled atmosphere–fire simulations. Benchmarking efforts reported in D3.1 were focused exclusively on this level, where a variety of different WRF pipelines were implemented and benchmarked on the VEGA EuroHPC system to identify the initial bottlenecks of the WRF-SFIRE code.

- Settlement level: At the finer urbanization scale, the pilot's purpose is to simulate flame-scale fire dynamics over built environments and complex terrains. This includes local atmospheric interactions, obstruction effects from buildings and vegetation, and the modelling of smoke and spark dispersion. The simulations leverage high-fidelity CFD solvers such as OpenFOAM and domain-specific fire modules. The OpenFOAM code for this level is currently under development and integration, so benchmarking is expected to be performed in the next part of the project.

3.4.1 Pilot progress and updates

For Deliverable D3.1, the WF pilot performed initial benchmarking activities focusing on landscape-scale wildfire simulations using the WRF-SFIRE model. These efforts provided early validation of the pilot's functionality on EuroHPC JU systems and highlighted several limitations. Firstly, the deployment process of WRF and WPS was hindered by system-specific software dependencies, making it difficult to port the pipeline across different HPC systems. Next, benchmarking was constrained to a set of baseline scenarios with limited scalability, focusing mainly on pipeline correctness and viability, not scalability. Consequently, the overall pipeline exhibited suboptimal scalability due to its complexity, decomposition limitations caused by the problem resolution, and I/O-heavy pre-processing/data loading phases. Consequently, in this deliverable the focus for the WF pilot was a) portability and b) evaluating, profiling and improving scalability in more complex WRF execution pipelines.

Table 18. Details of the Cadalso simulation case

Location	Domains	Horiz. Res.	Comp. Grid	Start point	Duration
Cadalso	4	5.4 km	640 x 642	28/06/2019 14:00 UTC	4 hours
		1.08 km	716 x 676		
		216 m	826 x 786		
		72 m	892 x 889		

Portability improvements

As part of the project's co-design efforts, the WF pilot is collaborating with the EPICURE project [24] to ease and standardise the installation and deployment of the

WRF-SFIRE modelling framework across multiple EuroHPC JU systems. The goal of this collaboration is to overcome the reproducibility and portability challenges highlighted in D3.1 by developing EasyBuild recipes and installation procedures for each HPC system. To this end, EPICURE has allocated dedicated personnel for each target platform, responsible for developing and validating installation workflows, which are subsequently tested by MTG in production environments.

As of now, the progress is as follows:

- LUMI: An EasyBuild recipe for WRF-SFIRE and WPS has been created and tested successfully.
- Discoverer: An installation procedure for WRF-SFIRE has been created and successfully tested. WPS installation is still pending.
- Vega: An EasyBuild recipe is currently under development and being tested.
- Meluxina: An EasyBuild recipe is currently under development
- Karolina: An EasyBuild recipe is currently under development.
- Mare-Nostrum: An EasyBuild recipe is currently under development and testing has begun.

New codes and pipeline changes

To explore scalability in a new 'larger' scenario with higher available parallel work, the WF pilot has expanded its technical scope to include higher-resolution simulations at the landscape level. To that end, a new WRF-SFIRE simulation was configured for the wildfire event that began on 28/06/2019 at 14:00 UTC near Cadalso de los Vidrios in Comunidad de Madrid. This case introduced a 72-meter horizontal resolution in the innermost domain, representing one of the highest-resolution wildfire simulations conducted within the WF pilot to date. Its configuration details are presented in Table 18. The simulation ran for a total of 4 hours, initialized at ignition time. Initial and boundary conditions were sourced from the ERA5 [25] reanalysis dataset at a resolution of 0.25°, while the digital elevation model and fuel model map were generated using LIDAR-derived raster layers provided by the National Geographic Institute.

The forest fuel classification followed the Anderson (1982) scheme, consistent with BEHAVE[26] modelling standards, and was processed to a 25-meter spatial resolution. Due to differences in grid structures between domains, the ndown.exe [27] utility was required for domain coupling between d02 and d03. The outer domains were executed on 32 nodes (128 cores per node), while the finer domains initially used ~40 nodes with identical core configurations to accommodate computational load and memory demands.

3.4.2 Performance analysis

D3.1 assessed the scalability of the WRF-based workflow (WF) for two scenarios and across different node configurations in VEGA in an attempt to identify its key performance limitations. The larger scenario ('2k_test', henceforth referred to as '200m-Robledo') employed there revealed that while speedup improved up to 10 nodes, it degraded beyond that point due to pre-processing and intermediate data handling becoming increasingly dominant as computational time decreases.

In this deliverable a different approach is used. First of all, simulation is redefined to denote only the final WRF stage instead of all the WRF stages and the Real stage as in D3.1. Second, all pipeline stages are analysed in the breakdowns plots, to identify specific bottlenecks for each one of them. The node counts displayed in all such plots refer to the final WRF stage, while preceding stages may use fewer nodes.

Table 19. Programming & runtime environment for WF-WRF benchmarks

	LUMI	Karolina	Leonardo
Compiler	GCC-13.2.0	GCC-13.2.0	GCC-12.2.0
Parallel framework	MPICH v8.1.29	Open MPI v4.1.6	
Libraries			
zlib	1.3.1	1.2.13	1.3
cuRL	-	8.3.0	8.4.0
HDF5	1.12.2	1.14	1.14
libpng	1.6.40	1.2.50	1.2.50
jasper	4.0.0	1.900.1	1.900.1
NetCDF-C	netcdf-hdf5parallel v4.9.0.11	4.9.2	4.9.2
NetCDF-FORTRAN	netcdf-hdf5parallel v4.9.0.11	4.6.0	4.6.0
WRF-SFIRE	4.4	4.4	4.4
WPS	4.4	4.4	4.4

WF – System configuration

Three EuroHPC JU systems have been used for WF's scalability analysis in addition to Vega used in D3.1: LUMI, Karolina, and Leonardo. The programming and runtime environments for WRF and WPS installation on each system are detailed in Table 19.

WF – Results & Analysis

First, we compare the performance of WF reported in D3.1 on Vega (AMD-based system) with the performance achieved for Leonardo (Intel-based system) for the same 200m-Robledo scenario. Besides comparing different CPU vendors, we also employ Leonardo as our new baseline for scalability for two reasons: i) It is a more recent

system, and ii) it allowed the deployment of the pilot without the technical issues that we experienced before on Vega. Figure 22 depicts the scalability achieved on Leonardo and LUMI. It is evident, that while Leonardo exhibits improved scalability compared to VEGA, speedup still diminishes beyond 16 nodes.

Figure 23 presents the detailed execution breakdown per stage for the 200m-Robledo pipeline on Leonardo, allowing the identification of two possible causes for this behaviour: i) The last WRF stage does not scale after ~16 nodes, and ii) the remaining pipeline stages do not scale at all and therefore quickly become a bottleneck.

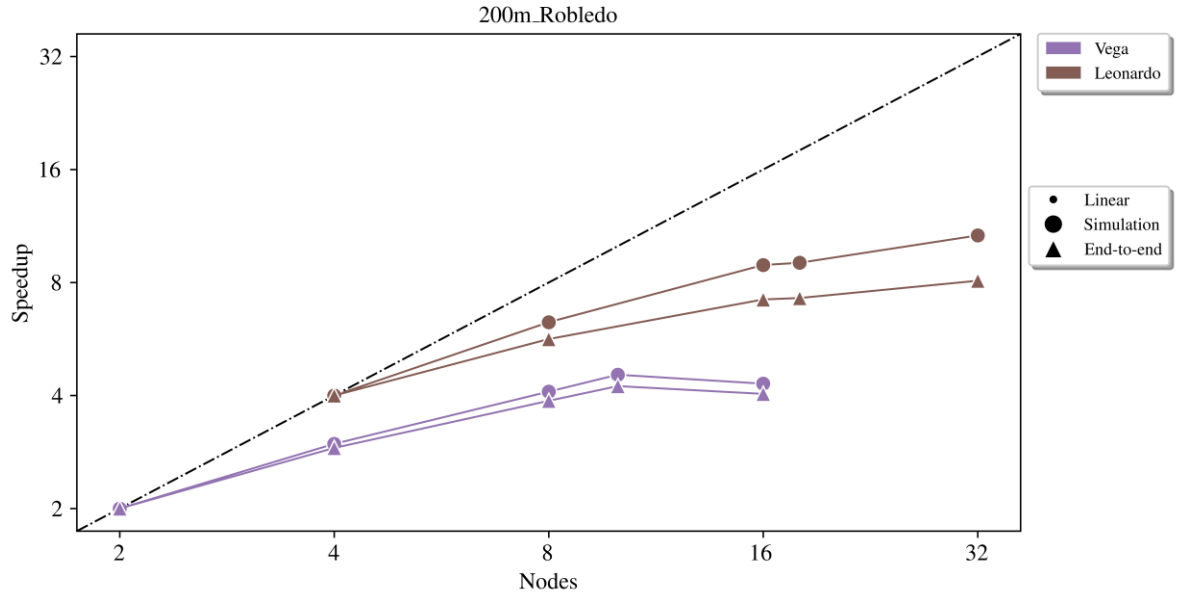


Figure 22. WF per node speedup on VEGA and LUMI for the 200m-Robledo scenario

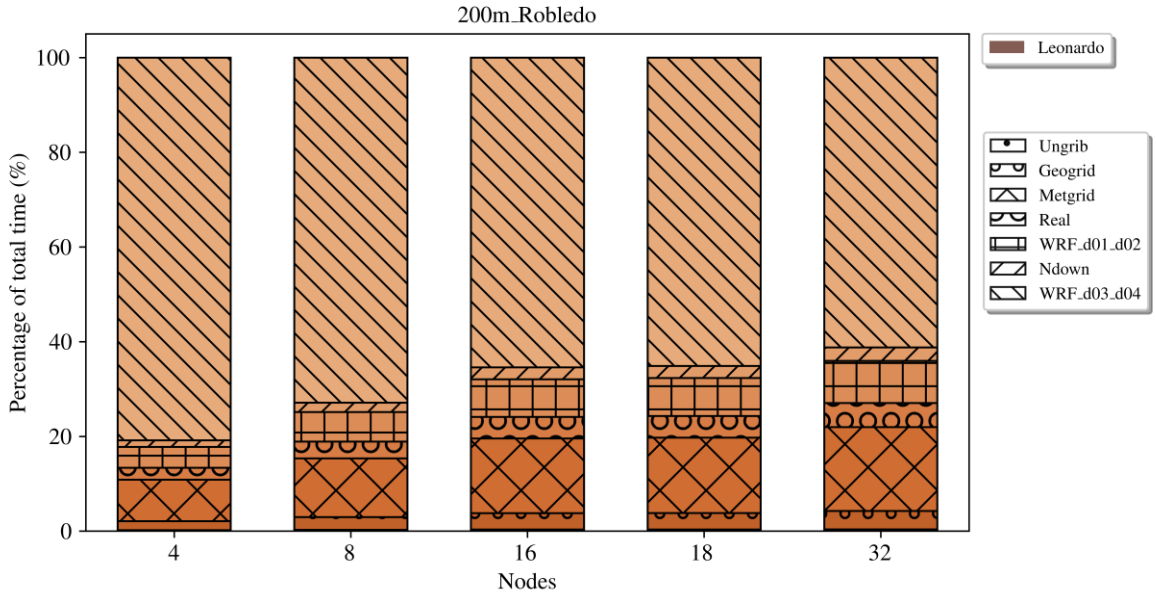


Figure 23. WF execution time breakdown per stage for 200m-Robledo in Leonardo

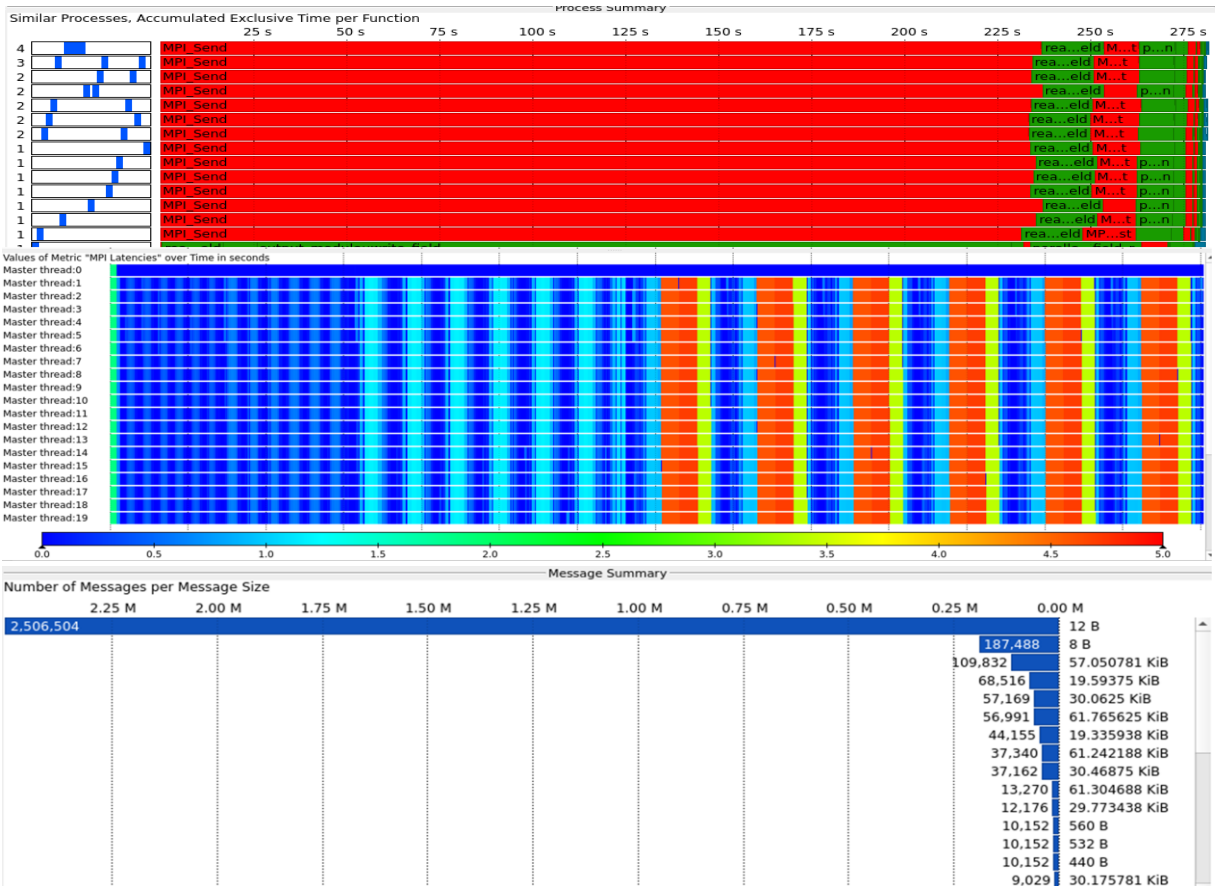


Figure 24. WF's Metgrid stage tracing for the 200m-Robledo scenario

The former is problem-specific, as the maximum scalability depends on the WRF domain decomposition, and for this pipeline does not enable the utilisation of more computing resources (nodes). The latter however constitutes a critical problem, as even for a more scalable WRF domain/problem, the rest of the stages will limit the performance as defined by Amdal's law. Hence, in order to address this issue and detect potential solutions, the pipeline components were profiled and traced with Score-P and visualized with Vampir [28], leading to the following observations:

As indicated in

- Figure 23, the **Metgrid** stage has a significant impact on the total execution time, accounting for nearly 20% of the end-to-end execution duration when WRF runs on 32 nodes. We note that this stage executes on a single node with 25 processes, exhibiting extremely low parallel efficiency (0.06). Figure 24 groups Metgrid processes by their communication/computation patterns, revealing that Metgrid is heavily communication-bound, with MPI_Send (mostly) and MPI_Broadcast comprising over 90% of total time. We can also observe

that some messages have very high latency. With further analysis, we attribute this to the exchange of many very small messages during execution. Consequently, this problem is caused by the Metgrid algorithm, which was not designed to be highly parallel and, therefore, scalable.

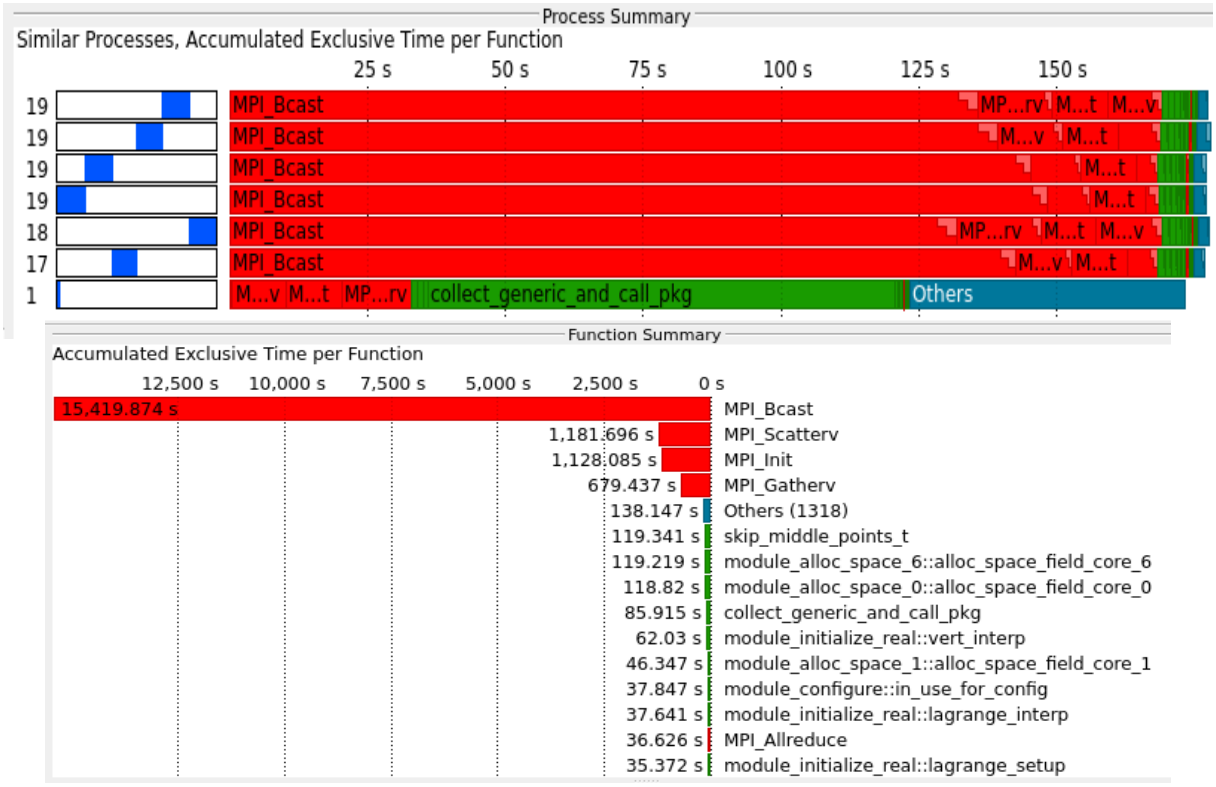


Figure 25. WF's Real stage tracing for the 200m-Robledo scenario

- The **Real** stage executes on 1 node with 112 processes and takes around 5% of the end-to-end execution time when WRF runs on 32 nodes. As shown in Figure 25, it suffers from imbalance, with process 0 performing some computation and all the other processes being dominated by MPI communication. MPI_Scatterv and MPI_Broadcast account for over 90% of execution time, resulting in an extremely low parallel efficiency (0.01). As a result, scalability is inherently constrained, and increasing process/node count does not improve performance.

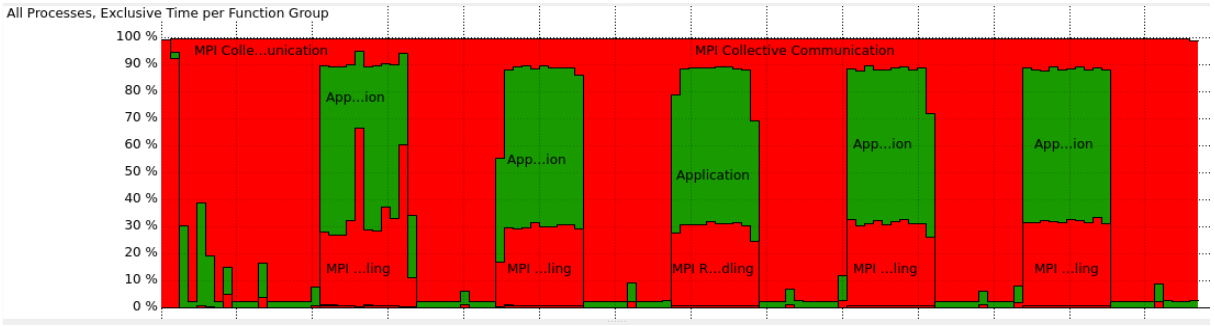


Figure 26. WRF_d01_d02_d03 stage tracing for the 200m-Robledo scenario

- The **WRF_d01_d02 stage** runs on 8 nodes (896 processes) and takes around 10% of the end-to-end execution time when WRF runs on 32 nodes. This stage is similar to the last WRF stage but operates on a smaller grid. It involves both computation and collective communication and has an estimated parallel efficiency of 0.33. Figure 26 outlines the process timeline and shows low computational imbalance, with 50% of execution time spent on MPI_Bcast collective communication between computation phases and 15% blocked by MPI_Wait and other dependencies.

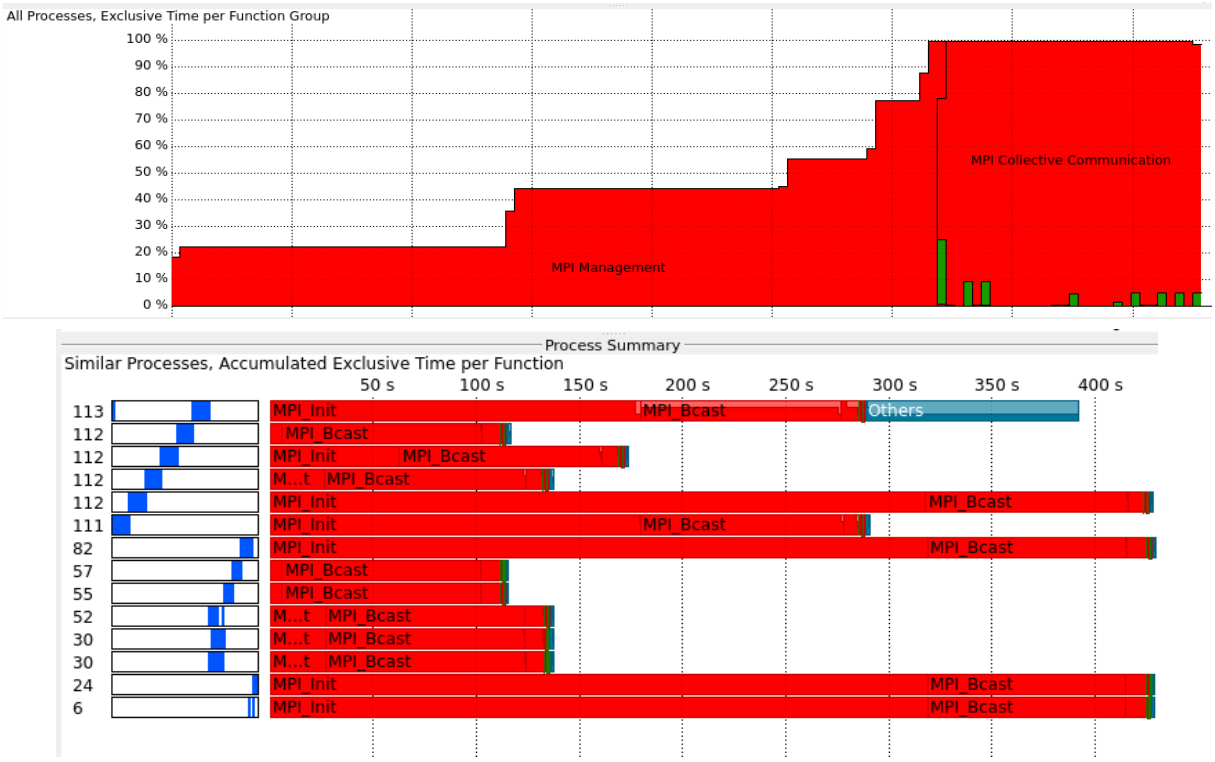


Figure 27. WF's Ndown stage tracing for the 200m-Robledo scenario

- The **Ndown** stage executes on 8 nodes and takes less than 5% of the end-to-end execution time when WRF runs on 32 nodes. As depicted in Figure 27,

25% of Ndown's time is spent on collective communication (MPI_Bcast), 5% on computation, and 70% in MPI management (MPI_Init), with processes spending 10–75% of their total time in MPI initialization and minimal computation, leading to Ndown exhibiting extremely low parallel efficiency (0.01).

- The final WRF stage accounts for 60–80% of total execution time while utilizing ~90% of total resources. It follows the same communication/computation pattern with the previous WRF stage but uses a larger decomposition grid, improving parallel work and therefore, scalability. It starts with a parallel efficiency of 0.78 at 8 nodes, which declines sharply as node count increases, dropping to ~0.3 at 32 nodes.

Summarizing, the WRF stages consume most of the execution time and resources. While they scale much better than the other stages, communication remains a limiting factor and scalability depends on the domain decomposition which defines the parallel work and the communication-to-computation ratio. But, even with perfect WRF scaling, the maximum achievable end-to-end speedup for 200m-Robledo would be ~40×, constrained by the rest stages that do not scale, with Metgrid being the main bottleneck.

Based on the profiling and analysis of the WRF 200m-Robledo pipeline, MTG is exploring hybrid implementation of WRF-SFIRE using MPI and OpenMP (previously only with MPI). This process is ongoing, with a hybrid version successfully compiled and validated with small-scale configurations on the Vega system, but with runtime errors appearing in larger simulations (when exceeding 900 grid points). This issue has been reported to the Vega support team and EPICURE and is currently under investigation. At the same time, EPICURE is also testing hybrid implementations on LUMI which will also utilize custom vectorisation optimizations (originally developed for Discoverer). Once hybrid MPI+OpenMP WRF passes the testing phrase, EPICURE has advised fine-tuning task placement, OpenMP usage across different subprograms, and usage of compilation flags. For example, initial results showed that Metgrid performed better when configured to use two OpenMP threads and half of the MPI tasks instead of the default numbers.

Table 20. Details of new WF Cadalso scenarios

	D01 (km)	D02 (km)	D03 (km)	D04 (m)	Number of points per domain	
					e_we	e_sn
200m_ Cadalso	9	3	1	200	D01=345 D02=688 D03=676 D04=846	D01=249 D02=490 D03=484 D04=786
72m_ Cadalso	5.4	1.08	0.216	72	D01=640 D02=716 D03=826 D04=892	D01=642 D02=676 D03=786 D04=889

In addition to the actual scalability bottlenecks of the WF code explored above, the previously benchmarked pipelines were also constrained by low maximum available parallelism due to not being designed for HPC execution (prior to this project). To that end, we consider two more fitting WRF scenarios/pipelines for further scalability analysis, using a new "Cadalso" dataset and with the domain characteristics described in Table 20. These new tests are similar to Robledo-200m, but after the first four pre-processing stages which remain the same, they decompose domains differently and use different resolutions. More specifically:

- For the 200m_Cadalso:
  - WRF is run for D01, D02 and D03.
  - Ndown is used to remap the output of D03 to input D04.
  - WRF is run for D04.
- For the 72m_Cadalso:
  - WRF is run for D01 and D02.
  - Ndown is used to remap the output of D02 to input D03.
  - WRF is run for D03 and D04.

The 200m_Cadalso test was evaluated across all three new systems: LUMI (using 4 – 64 nodes), Karolina (using 1–128 nodes), and Leonardo (using 1–128 nodes). On the other hand, the 72m_Cadalso test was ran only on the Karolina (using 2 – 64 nodes) due to the limited resources of the EuroHPC JU benchmarking access scheme.

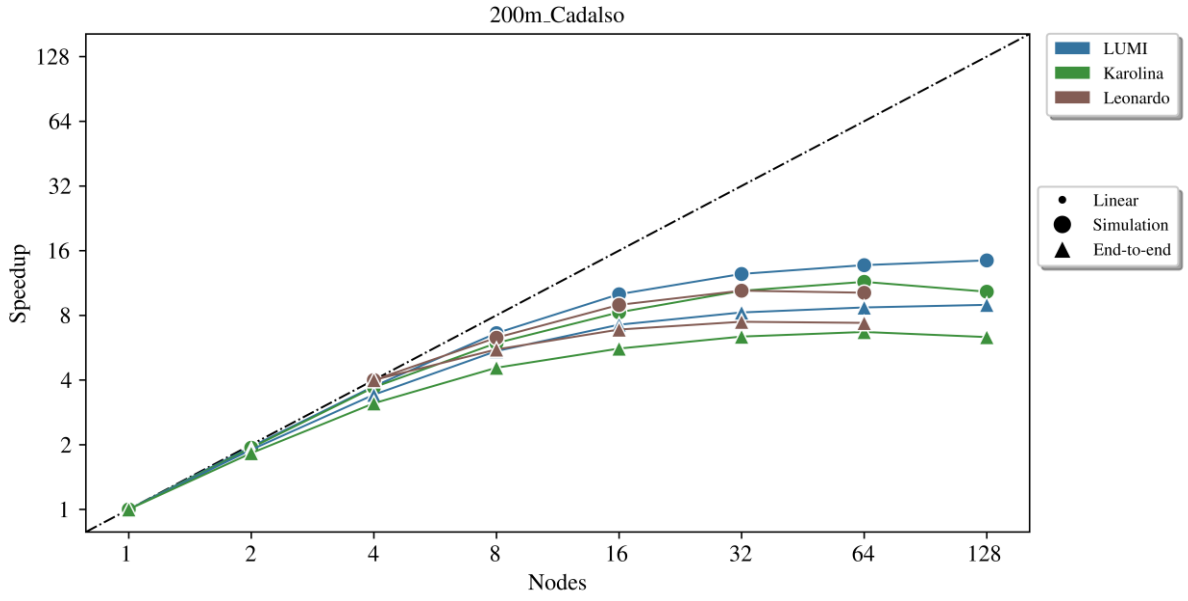


Figure 28. WF per-node speedup for the 200m_Cadalso scenario

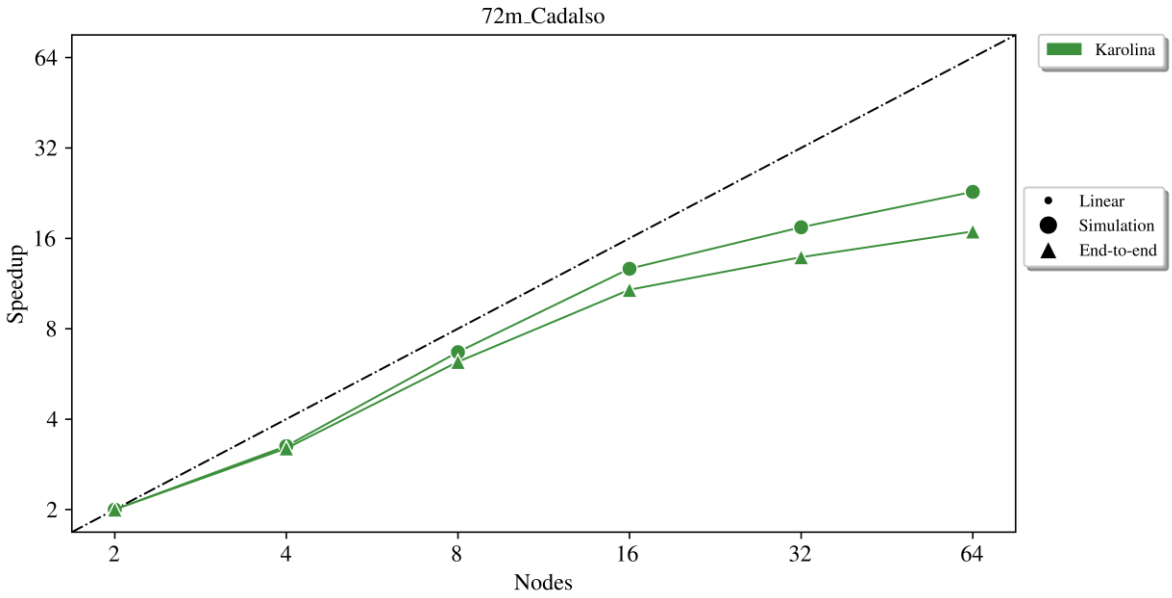


Figure 29. WF per-node speedup for the 72m_Cadalso scenario

Figure 27 and Figure 29 present the achieved speedup for both the end-to-end execution and the final WRF simulation stage. Both tests demonstrate superior scalability to 200m_Robledo, due to the larger amount of computation in the last WRF part. The simulation of the 200m_Cadalso scenario scales well up to 8 nodes (~6x speedup) for all three systems and then shows a minor performance gain until 64 nodes, albeit at a much lower scaling rate (~11x speedup). However, overall scalability remains limited for end-to-end execution due to the increasing overhead of the other stages, achieving a maximum end-to-end speedup of 8.95x on LUMI (lower on the other systems). On the other hand, the 72m_Cadalso scenario exhibits higher scalability achieving almost double speedup for in 64 nodes (~23x) compared to 200m_Cadalso due to the further increased parallel work because of the higher resolution. Finally, its end-to-end speedup follows a similar trend (~16x for 64 nodes) but still becomes visibly constrained by the other stages after ~8 nodes.

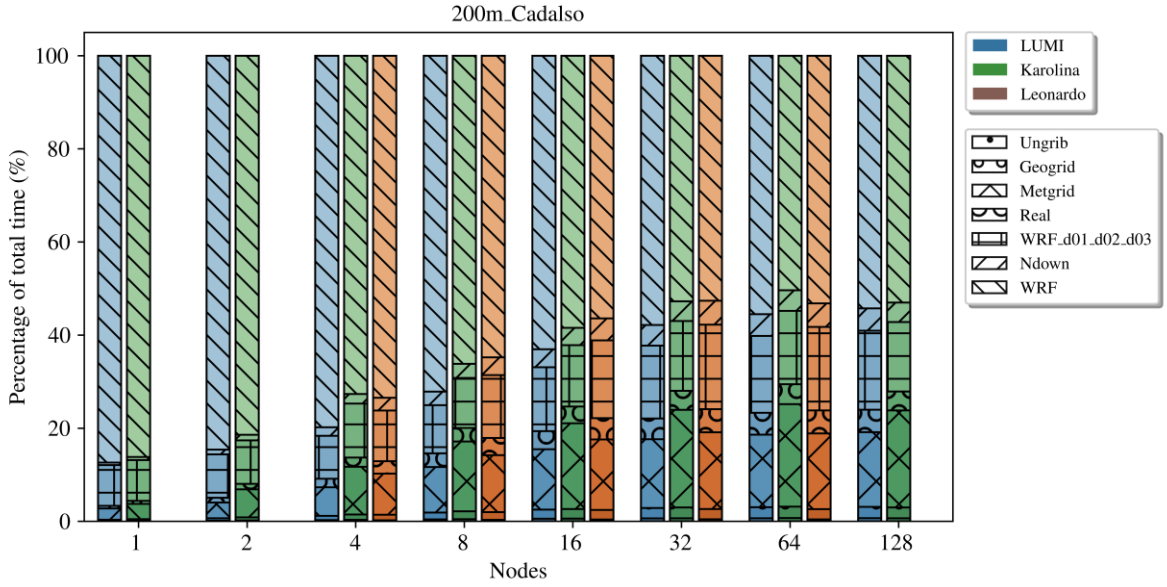


Figure 30. WF's execution time breakdown per stage for the 200m_Cadalso scenario

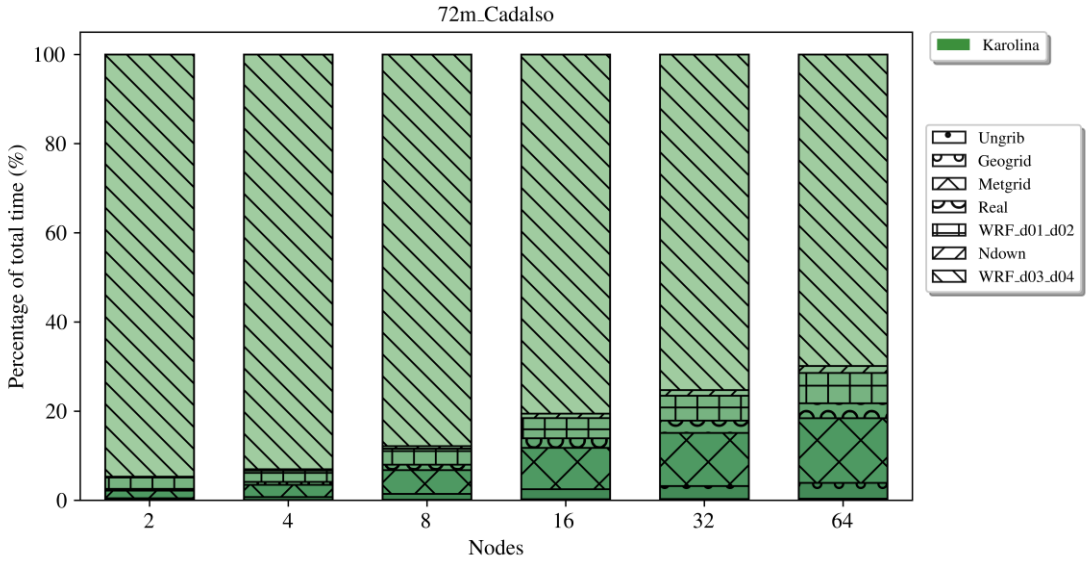


Figure 31. WF's execution time breakdown per stage for the 72m_Cadalso scenario

These observations are confirmed by the execution time breakdown depicted in Figure 30 and

Figure 31 for the 200m_Cadalso and 72m_Cadalso scenario respectively. As shown in Figure 30, Metgrid and the first WRF stage (D01, D02, D03 computation) constitute the primary bottlenecks similarly to the previously discussed 200m_Robledo scenario. More specifically, the simulation dominates for ≤16 nodes (occupies >50% of total execution time), but beyond 16 nodes the speedup flattens and pre-processing and simulation contribute equally (~50%) to the total time.

In contrast, the 72m_Cadalso scenario exhibits a different execution profile. As shown in Figure 29, the final WRF stage still dominates total execution time even for higher node counts, starting from 95% of total execution time for 1 node and ending up to ~70% for 64 nodes. Additionally, the first WRF stage, comprising in this case only of two domains (the third has been shifted in the final WRF invocation), now contributes significantly less to total time. On the other hand, Metgrid still emerges as a bottleneck at high node counts.

WF - Summary and next steps

Concluding, the WF pilot undertook during the reporting period following D3.1 a series of corrective actions aimed at improving portability and scalability. In respect to portability, with support from the EPICURE co-design task, the pilot's build system and execution environment were refactored to ease deployment on new EuroHPC systems, reducing hard dependencies and improving modularity. In respect to scalability, the benchmarking suite was extended with more diverse and demanding scenarios that better exercise the scalability of the pipeline and stress the system's computational resources. In order to address the previously observed scalability bottlenecks, in-depth profiling was performed using Score-P and analysed with Vampir [28]. This analysis provided detailed insights into performance breakdowns across the pipeline's stages, highlighting opportunities for parallelism improvements, I/O restructuring, and load balancing.

The next step for the WF pilot is integrating and evaluating potential WRF scalability improvements. To that end, WF will evaluate a hybrid MPI+OpenMP WRF-SFIRE implementation, which is currently under development and early tests on Vega have shown promising results on small-scale configurations. Additionally, hybrid builds with vectorisation optimisations for WRF-SFIRE will be tested on LUMI. Once code correctness is confirmed, the focus will shift to tuning OpenMP settings, task placement, and compilation options to further improve scalability and performance. Finally, the WF will focus on the benchmarking, evaluation and optimization of the new OpenFOAM code currently under development.

3.5 Material Transport in Water (MTW)

The **Material Transport in Water (MTW)** pilot use-case is designed to simulate the complex interactions between fluid dynamics, particulate matter and temperature changes. As rising water temperatures threaten aquatic ecosystems, especially rivers and oceans, accurate computational models and extreme-scale simulations are critical to studying climate change and predicting its consequences for aquatic life.

To tackle this challenge, MTW leverages the **waLBerla** [29] [30], a high-performance multiphysics framework based on the **Lattice Boltzmann Method (LBM)**. LBM allows for scalable simulations that efficiently handle coupled fluid-thermal dynamics, essential for studying material transport in water systems.

3.5.1 Pilot code description

FAU officially joined the HiDALGO2 project in **M18 (June 2024)**, i.e., after the submission of Deliverable D3.1. This late integration required a rapid adaptation of the existing waLBerla workflows to fully utilize the capabilities of EuroHPC systems efficiently while maintaining the flexibility needed to model complex environmental scenarios. Consequently, in this deliverable the MTW pilot focuses on: i) adjusting to HIDALGO2 methodology, ii) porting existing code to EuroHPC machines, and iii) establishing an optimization baseline for the project.

MTW – Adjusting to the HIDALGO2 workflow

The benchmarking workflow for the MTW pilot at FAU follows a well-known Research Data Management (RDM) framework to ensure efficiency and reproducibility in performance evaluation. It relies on three primary repositories: *benchmark-scripts* for execution scripts, *benchmark-data* for performance data storage, and *post-processing* for data analysis and visualization (details about FAU's RDM can be found in D2.5). Additionally, to integrate with the HiDALGO2 benchmarking framework a project-specific repository *hid-bench-mtw* has been introduced to the workflow.

The current process consists of cloning the benchmark-scripts repository, where dedicated build scripts for each cluster and application are responsible for software retrieval and compilation. Then, batch run scripts execute benchmarks using internal waLBerla timers to collect performance metrics, which are then uploaded to benchmark-data. Post-processing scripts convert this structured performance data into CSV format for hid-bench-mtw, ensuring it aligns with HiDALGO2 methodology requirements. Current transformations can be done effectively without relying on ReFRAME [31]. Integrating ReFRAME into the MTW pilot has proven to be more complex than the current MTW workflow requires. As a result, its integration has been deprioritised in order to maintain consistency and efficiency with established benchmarking practices. However, considering ReFRAME's automated environment matching capabilities, a ReFRAME integration into future workflows will be considered, especially if advanced benchmarking pipelines with increased modularity are required.

MTW – Code optimizations prior to HiDALGO2 entry

The parallelization of the MTW pilot is based on **waLBerla,** an HPC framework designed for scalability and portability [32] [33]. The key component driving these optimizations is **lbmpy** [34] [35] [36], a code-generation framework that combines the

high-level mathematical description of LBM models and efficient, low-level implementations suited for diverse hardware architectures.

Specific optimizations include:

- **Common Sub-expression Elimination (CSE)** in order to reduce redundant computations.
- **SIMD Vectorization** for CPU-based parallelization (SSE, AVX, AVX512).
- **Memory packing** and efficient **in-place streaming patterns** to ensure optimal use of memory bandwidth *(not applied to the MTW pilot yet).*
- **Hardware-specific code generation** that allows automatic adaptation to CPUs and GPUs.
- **Communication overlap/hiding** by splitting the problem domain into an inner and outer layer for every process. This allows the parallel update of the local inner cells while communication of the outer ones is performed, reducing the idling time spent while waiting for the massage-based communication to finish *(not applied to the MTW pilot yet).*

In this deliverable, the MTW pilot focuses on two main benchmark test cases:

- **Fluid Benchmark – Lid-Driven Cavity Flow** (henceforth "*UniformGrid*"): This benchmark focuses on the well-known lid-driven cavity test case, where shear-driven flow is induced within a closed square cavity. The simulation is implemented using LBM, specifically evaluating the performance of D3Q19 and D3Q27 models with the Single Relaxation Time (SRT) collision scheme. Each timestep involves fluid updates, boundary condition enforcement, and inter-processor communication through MPI-based ghost layer exchanges, enabling scalable parallel performance assessment. The exact benchmarking setup can be found in Table 22.
- **Two-Way Coupled Fluid-Temperature Benchmark – Differentially Heated Cavity** (henceforth "*MTW-case*"): This test case simulates thermally induced convection within a fluid domain using a coupled lattice Boltzmann approach. It combines two LBM models, each applied at every simulation timestep with two-way interaction: a D3Q19 model with a single relaxation time (SRT) scheme for fluid dynamics, and a D3Q7 model with a multiple relaxation time (MRT) scheme for temperature evolution. The simulation involves fluid and temperature field updates via LBM, appropriate boundary handling to preserve coupling consistency, and ghost layer communication of particle distribution functions (PDFs) for both fields. This ensures efficient data exchange across distributed memory systems, supporting scalable parallel execution. From a pure computational point of view, the MTW case is the successive execution of two LB kernels with different boundary conditions and memory array sizes.

Therefore, the performance optimizations presented for the pure fluid kernel should in theory be directly applicable to the MTW showcase. However, as the mutual coupling of these two properties introduces additional complexity, this remains an optimistic assumption for now and thorough benchmarking and validation is required. The exact benchmarking setup can be found in Table 22 and Table 23.

3.5.2 Co-Design Activities

Due to the recent addition of FAU to HIDALGO2, efforts in profiling and optimizing the pilot for a specific target EuroHPC system have not started yet. Consequently, co-design activities performed by FAU for the moment are executed on the abstraction layer of code-generation. The key co-design activities explored in the context of the HIDALGO2 project are:

- Optimizing memory access patterns for high-bandwidth memory (HBM) configurations.
- Efficient inter-node communication using waLBerla's MPI-based implementation, ensuring scalable distributed simulations.
- GPU offloading with lbmpy- and pystencils [37]-generated computational kernels, which are specifically optimized for different GPU architectures. The generated kernels efficiently execute on:
  - NVIDIA GPUs via CUDA-generated kernels
  - AMD GPUs via HIP-generated kernels
- Testing the performance characteristics of waLBerla for various architectures on a local FAU test cluster. This led to integrating waLBerla with **lbmpy-generated LBM kernels** against different hardware platforms for continuous optimization [38].

3.5.3 Performance analysis

This section provides a performance evaluation of the MTW showcase through an investigation of the pure fluid benchmark "UniformGrid" with all of waLBerla's optimization strategies available as detailed previously and the fluid-concentration-coupled benchmark "MTW case". The analysis is based on strong scaling and focuses on: i) computational speedup for both benchmark applications, and ii) the distribution of execution time among the various simulation components of the "MTW case", in line with the HIDALGO2 benchmarking and reporting methodology established in D3.1. While the waLBerla framework is able to extensively optimize the LBM simulations present in "UniformGrid", the LBM model of the transport equation as well as the fluid-temperature-coupling which are utilized by the "MTW case" remain in an exploratory

phase, where fewer optimizations have been implemented to ensure a comprehensive assessment of workload distribution and correctness.

MTW – System configuration

MTW executions were performed during the reporting period on two EuroHPC systems, LUMI and MareNostrum5, on both the CPU and GPU partitions. The details of the programming and runtime environments used for these runs can be found in Table 21. The compilers were chosen based on their reputation to achieve optimal performance on target architecture. All reported benchmarks use the most recent library versions available on the systems.

MTW – Benchmarking configuration

The UniformGrid benchmark was executed on two GPU based and two CPU based systems with a Lattice model consisting of 19 and 27 PDEs per cell respectively. These systems represent the different benchmark scenarios that can be found in Table 22. The in-place AA streaming pattern with collision of single relaxation time are exemplarily chosen. Each benchmark was set up to fit the problem in the respective CPU or GPU memory of a single node resulting in given distributions of "Lattice Cells per Process". Due to the benchmark's simplicity, the performed number of time-steps was kept minimal with a pure simulation time of approximately 3 seconds on one node. The reduced run time during strong scaling was compensated by repeating the simulation proportionally to the number of utilized nodes and averaging over the simulation time per run.

Table 21. Programming & runtime environment for MTW benchmarks

	LUMI-C	LUMI-G	MN5-GPP	MN5-ACC
Compiler	Cray-clang-17.0.1	AMD-clang-17.0.0	Intel-2021.10.0	NVHPC-23.11.0
Parallel framework	cray-mpich-8.1.29	cray-mpich-8.1.29	Openmpi-4.1.5	Openmpi-4.1.5
Libraries				
waLBerla	6.1	6.1	6.1	6.1
lbmpy	1.3.7	1.3.7	1.3.7	1.3.7
Python3	3.11.7	3.11.7	3.12.1	3.12.1
CUDA	-	-	-	12.2
HIP	-	6.0.3	-	-
Jinja2	3.1.6	3.1.6	3.1.6	3.1.6
Pybind11	2.13.6	2.13.6	2.13.6	2.13.6

Table 22. Benchmarking configuration for the MTW-UnformGrid benchmarks

Scenario	Lattice Model	Communication Hiding	Lattice Cells per Process	Time-steps
MN5-GPP	D3Q27-AA-SRT	simple overlap with (8, 8, 8)-split	224x224x112	5
LUMI-C	D3Q27-AA-SRT	simple overlap with (8, 8, 8)-split	192x192x160	5
MN5-ACC	D3Q19-AA-SRT	simple overlap with (8, 8, 8)-split	640x640x640	20
LUMI-G	D3Q19-AA-SRT	simple overlap with (8, 8, 8)-split	640x640x640	20

The *UniformGrid* benchmark utilizes all performance optimizations that are currently available within waLBerla and lbmpy. This includes architecture-specific optimizations produced by lbmpy, i.e. vectorization, loop transformations and memory packing, in-place streaming pattern, etc, and communication-hiding techniques of the waLBerla MPI module to overlap computation and communication by splitting the process local domain as specified in the respective column of Table 22. These optimizations enable the benchmark to maintain a high level of efficiency across multiple nodes.

The configuration data for MTW-case can be found in Table 23. Both the fluid and the temperature model use a pull streaming pattern instead of a more complex in-place one. Both cases are expected to have reached a steady state after the number of time steps specified.

Table 23. Benchmarking configuration for the MTW-case benchmarks

Scenario	Lattice Model fluid	Lattice Model temperature	Lattice Cells per Process	Time-steps
full_mem	D3Q19-pull-SRT	D3Q7-pull-MRT	256x512x128	1000
small_mem	D3Q19-pull-SRT	D3Q7-pull-MRT	128x256x64	1000

MTW – Results & analysis

Initially, strong scaling benchmarks for UniformGrid were carried out on two GPU-based systems: MareNostrum5-ACC and LUMI-G. First, GPU-focused benchmarking was conducted on MareNostrum5-ACC, reaching up to 32 nodes (i.e. 128 H100 NVIDIA GPUs) to assess strong scaling for the most recent NVIDIA Hopper architecture. Then, strong scaling experiments extended up to 512 nodes (2048 MI250x AMD GPUs) in LUMI-G to provide performance insights at extreme scale.

Figure 32 shows the strong scaling performance for UniformGrid with walBerla and lbmpy on the two GPU clusters LUMI-G (blue) and MareNostrum-ACC (green). The UniformGrid benchmark shows almost linear scaling up to 32 nodes. As the number of nodes increases, the scaling behaviour becomes logarithmic, which is still an adequate behaviour for strong scaling, especially on GPUs.

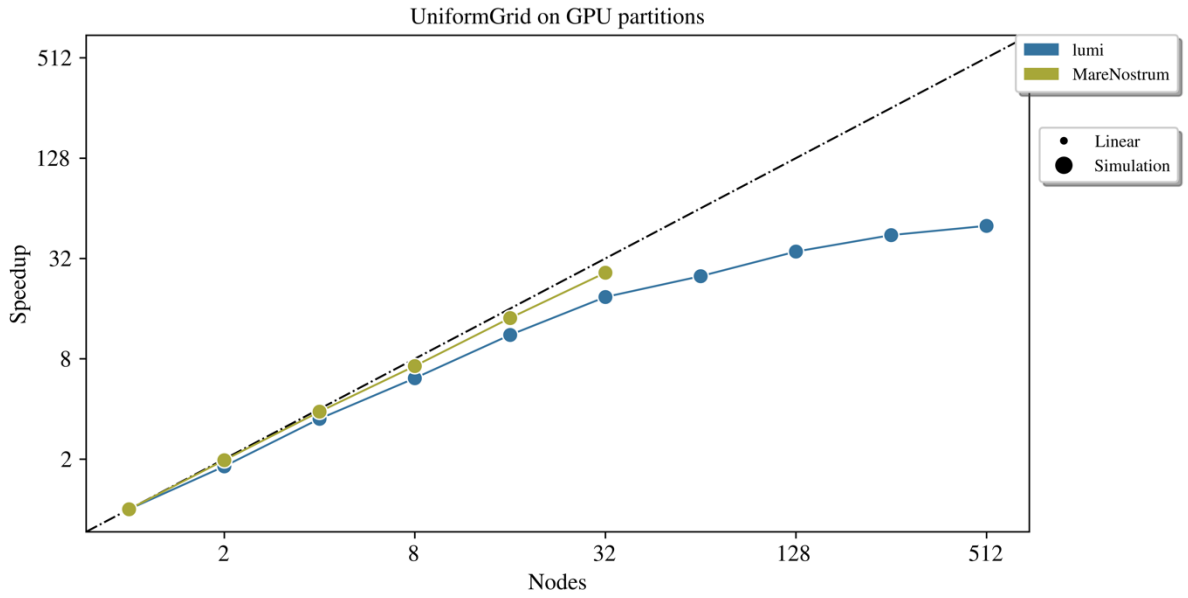


Figure 32. MTW per node speedup for strong scaling investigations of the fully optimized pure fluid LBM benchmark (UniformGrid) on LUMI-G (AMD) and MareNostrum5-ACC (NVIDIA)

In addition to GPU benchmarks, strong scaling analysis for UniformGrid was also carried out on two CPU-based systems: MareNostrum5-GPP and LUMI-C. On MareNostrum5-GPP, tests were performed using up to 64 nodes (~7k Intel Sapphire Rapid Cores), while on LUMI-C scaling was evaluated up to 512 nodes (~65k AMD EPYC Cores). Additionally, an attempt was made for running on Deucalion, but benchmarking was delayed due to MPI compatibility issues and challenges in achieving reliable cross-platform compilation.

As Figure 33 shows, UniformGrid exhibits near-optimal scalability across the full range of available nodes both for LUMI-C and MareNostrum-GPP, with two noticeable anomalies. First, there is a transient drop in scalability observed at 32 nodes on MareNostrum-GPP. Second, starting at 128 nodes on LUMI-C, the speedup increases over-proportionally with the number of nodes. We attribute both observations to the caching effects inherent to each CPU architecture. Furthermore, because CPUs act as general-purpose computing units, they can deliver robust performance even when dealing with smaller problem sizes. This behaviour contrasts with that of GPUs, which are designed for high-throughput computation. As a result, CPUs can manage lower

computational loads and communication of smaller data packets more effectively compared to GPUs. This helps to reduce the impact of latency, which often affects scalability when deploying large computing resources.

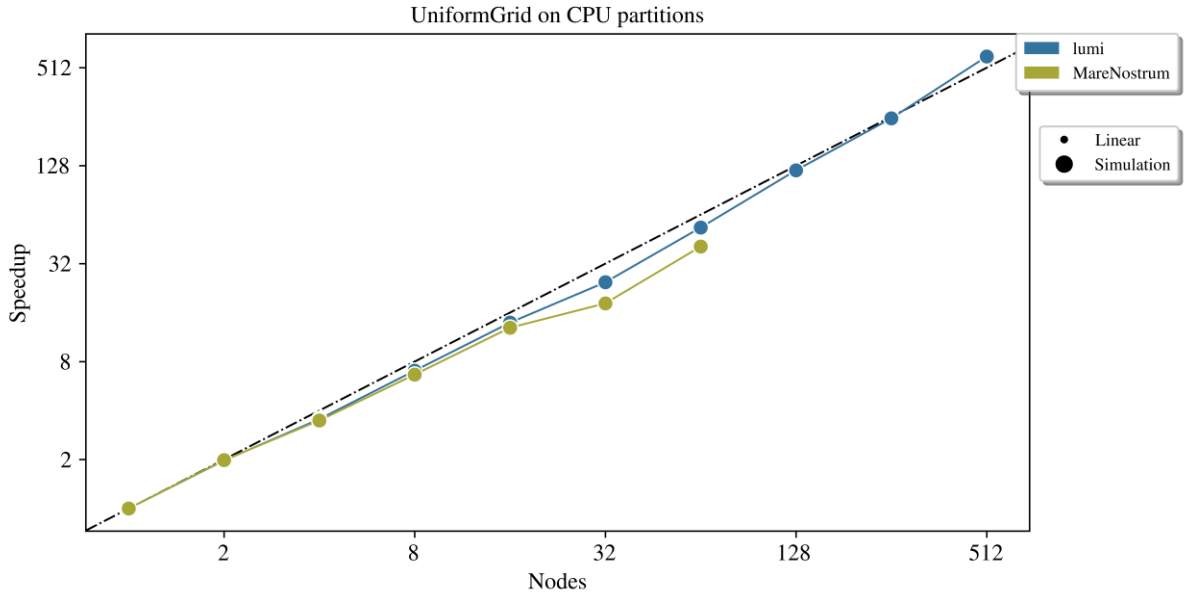


Figure 33. MTW per node speedup for strong scaling investigations of the fully optimized pure fluid LBM benchmark (UniformGrid) on LUMI-C (AMD) and MareNostrum5-GPP (Intel)

The MTW-case benchmarking focused primarily on LUMI-G, where strong scaling experiments were conducted using up to 128 nodes (each equipped with four AMD MI250x GPU modules). Two distinct problems (set up as detailed in Table 23) were evaluated: a *full memory configuration (light blue)*, in which the problem was sized to fully occupy the GPU memories of a single node, and *a small memory configuration (dark blue)*, where the problem size was reduced by a factor of two in each spatial dimension, resulting in a memory footprint approximately equal to one-eighth of the total GPU memory per node. The MTW-case was also deployed on Vega, but prolonged allocation times and resource contention prevented successful execution within the reporting period.

Figure 34 illustrates the logarithmic speed-up behaviour of the *full memory* configuration. The code displays some scalability up to 16 nodes, but after that point scalability collapses entirely. The *small memory* configuration follows a comparable trend but with scalability diminishing faster, which is expected considering that its initial problem size corresponds to one-eighth of the full memory setup. The poor scalability behaviour of this showcase can be traced back to exponentially decreasing utilization of the GPUs. When the number of nodes doubles, each node processes half as much data, resulting in many small communications between the GPUs, making the simulation latency-bound and under-utilizing the high GPU memory bandwidth. The

same applies to the computational power of the target architecture - the exponentially decreasing number of lattice cells per node has a direct impact on the number of threads available per compute unit, limiting the occupancy of the GPUs.

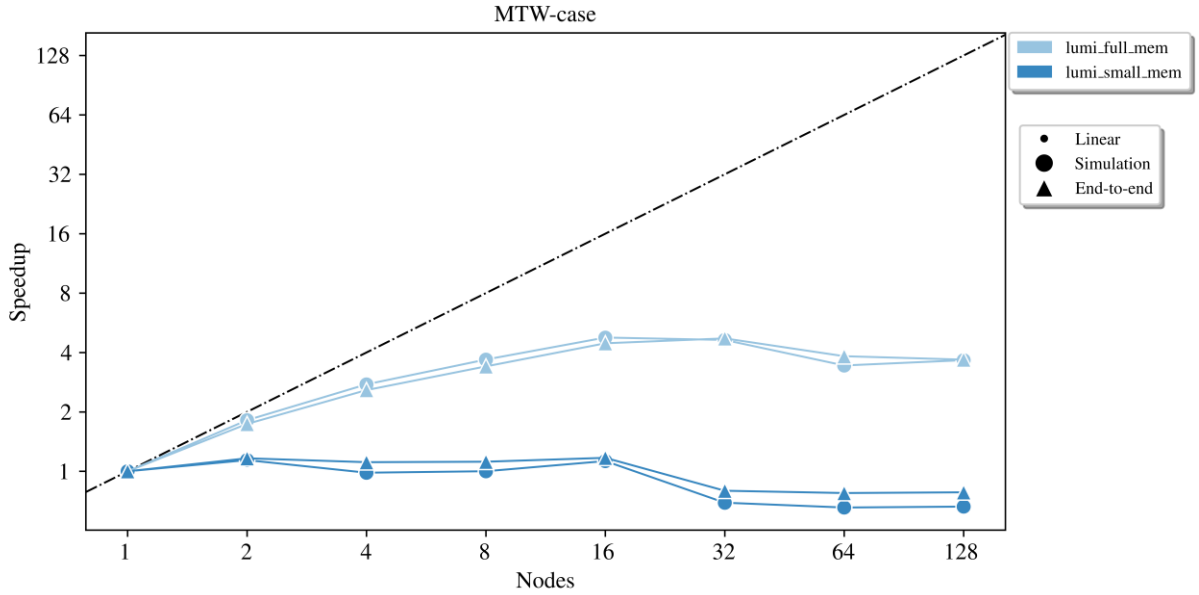


Figure 34. MTW per node speedup for strong scaling investigations of the MTW fluid-temperature-coupling's benchmark (MTW-case) on LUMI-G

To gain further insights into the bottlenecks of MTW-case, its execution is split into five steps:

- **Pre-Processing** that includes MPI initialization and memory allocation.
- **Fluid Communication** that includes inter- and intra-node MPI communication.
- **Fluid Update** that includes the stream-collide-step for approximating the fluids dynamic properties by solving the Boltzmann Equation, and the boundary handling.
- **Concentration Communication** that includes inter- and intra-node MPI communication.
- **Concentration Update** that includes the stream-collide-step to solve the transport equation for temperature as a scalar field using the LBM, and the boundary handling.

Figure 35 illustrates the percentile breakdown of the MTW-case execution to these stages, from bottom to top. The dominant influence of bandwidth and latency is evident; the higher the number of nodes, the lower the influence of the computation of the lattice cell updates to the point where the simulation just waits for the communication to finish. As LBM is a memory-bound algorithm, the computation times are directly proportional to the complexity of the LBM model, which can be seen well in the single node bar in Figure 35, where the fluid kernels with the Q19 model take

about 2.5 times as long as the temperature kernels with the Q7 model. On the other hand, the runtime impact of pre-processing tends to be constant within strong scaling.

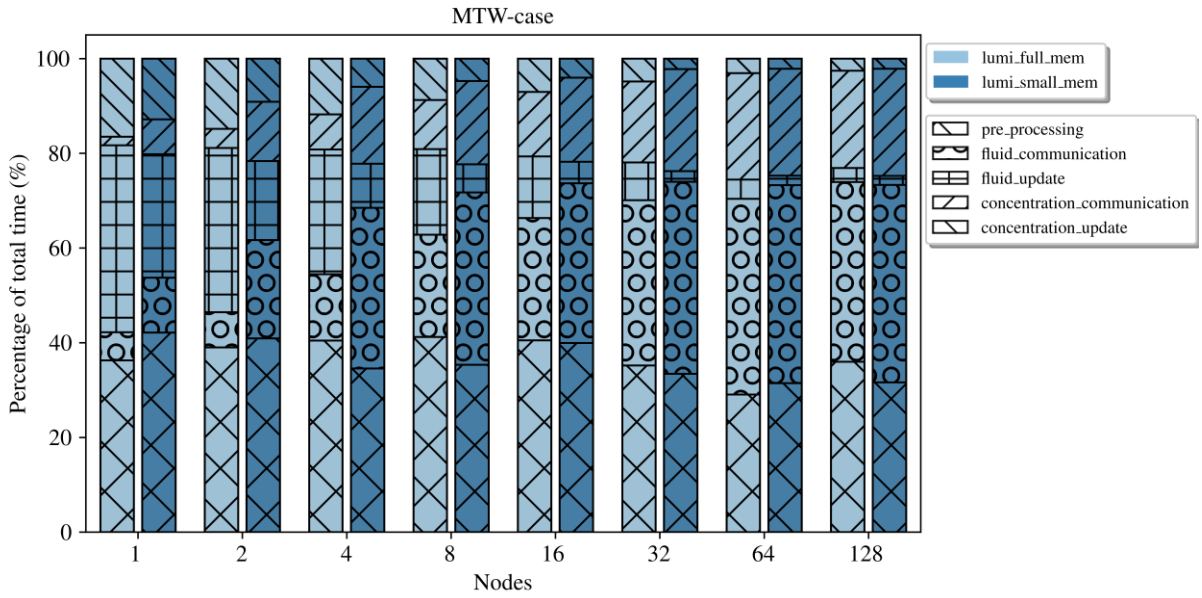


Figure 35. MTW-case benchmark's execution time breakdown on LUMI-G

MTW – Summary and next steps

Summing up the performance observations for the MTW pilot, while the Lattice Boltzmann Method demonstrates a significant degree of scalability, problem sizes per node must be sufficiently large to fully exploit node capabilities - especially on GPU clusters. In the subsequent deliverable D3.3, this should be further explored via weak scaling benchmarks and profiling. Furthermore, this analysis will be expanded to multiple EuroHPC systems, with the objective of validating performance and ensuring compatibility and efficiency across the entire EuroHPC hardware stack under production-level workloads. Finally, the exploration of task-level parallelism, such as ensemble simulations [39], could be regarded as a viable alternative to exclusive data-parallel methodologies.

4 Scalability & Optimisation related KPIs

HiDALGO2 has identified multiple KPIs related to benchmarking and optimisation activities and is committed to achieve ambitious targets. Table 24 presents the current status of these KPIs based on the benchmarking activities that have been reported in Section 3, along with the status reported on the previous deliverable (D3.1). It is evident that during the reporting period, good progress has been achieved.

Table 24. Status of benchmarking and optimisation related KPIs in M12 (D3.1) and M28

KPI	Target	M12	M28	Comments
Applications with a scalability of 50k cores in a single run	≥ 3	1	3	Scalable runs with more than 50k cores have already been achieved by UAP-Xyst, UAP-FOAM and MTW-walBerla.
Applications with a scalability of 200k cores in a single run	≥ 1	0	1	UAP-Xyst scales up to 196k processes in LUMI-C
Applications with a scalability of 80k cores in ensemble runs	≥ 3	0	0	No work has been performed yet for ensemble runs.
Applications with parallel efficiency improved by 30%	≥ 3	0	3	The parallel efficiency of WF-WRF code, UAP-Xyst and UAP-RedSIM has been improved

During the reporting period (M13-M28), HiDALGO2 has continued the benchmarking of its pilots on the HPC infrastructure, employing also the more recently added EuroHPC JU systems that were unavailable for D3.1. More specifically:

- The **RES** pilot extended its *damages* benchmarking activities to the Proxima and Leonardo systems, with scalability tests performed using up to 8k cores, doubling the core count reported in D3.1. A new scenario focusing on the prediction of *photovoltaic* energy production was introduced and benchmarked on Leonardo for up to 8k cores. Profiling of the *damages* workflow confirmed that communication remains the main bottleneck and future work will explore communication optimisations, such as communication-computation overlap.
- Benchmarking activities for the **UAP** pilot covered both **UAP-FOAM** and **UAP-Xyst** components on LUMI, with **UAP-FOAM** extending tests up to 512 nodes (64k cores). Scalability remained strong up to 128 nodes, with communication overhead becoming more significant at larger scales. **For UAP-Xyst**, three continuous Galerkin Finite Element solvers were evaluated, with successful **hero runs** on LUMI reaching 196k cores for LohCG and RieCG. In addition, **UAP-RedSIM** was tested on Karolina, scaling up to 64 CPU nodes (8k cores) and up to 32 NVIDIA A100 GPUs.

- The **UB** pilot underwent a major model redesign and was benchmarked on Discoverer, Karolina, MeluXina and VEGA, reaching up to 50 nodes (~6k cores). While scalability improved, issues with pre-processing and post-processing phases still persist at high node counts. Upcoming work will focus on file I/O optimisations to address these bottlenecks in the next deliverable.
- The **WF** pilot was benchmarked for D3.1on VEGA, with allocations of up to 16 nodes. Since then, extensive profiling and performance analysis has been carried out, revealing several bottlenecks in the pipeline, which are now under active improvement. The WRF-based pipeline has been extended to utilise 128 nodes (16k cores), with some scaling up to 64 nodes (8k cores). A new simulation scenario using OpenFOAM has also been tested and validated, and is expected to be benchmarked in the next phase.
- The **MTW** pilot joined the project after D3.1 and has adapted to the benchmarking methodology used across pilots, with ReFRAME integration underway. Benchmarks were carried out for both the standalone LBM kernel and the full MTW pipeline, on CPU and GPU architectures. The GPU-based LBM benchmark scaled effectively up to 2048 AMD MI250x GPUs on LUMI-G and 128 NVIDIA H100 GPUs on MN5-ACC. CPU-based benchmarks showed nearly linear scaling, reaching 512 nodes (64k cores) on LUMI-C. The full MTW workflow was tested on LUMI-G up to 128 nodes; however, scalability was limited and needs to be further analysed and improved.

In the forthcoming periods of the project, the benchmarking of the pilots will continue as they advance and are further optimised. The main next steps can be summarised as follows:

- ***More profiling and bottleneck analysis***: This second deliverable has already identified a few performance issues for pilot code, especially when scaling in larger amounts of nodes, using profiling and tracing along with usual benchmarking.
- ***Automating deployment:*** Due to differences between EuroHPC systems in terms of software dependencies, portability is one of the major issues still faced by pilots. More general solutions like containerization will be explored in the rest of the project.
- ***Extreme scalability:*** Future benchmarking activities will focus on scaling pilots to larger node counts and extreme-scale configurations, building on the progress achieved so far. As node counts increase, communication and I/O bottlenecks are expected to become more critical, and addressing these issues will be a key priority in the next phase of the project.

5 Conclusions

This deliverable reports the outcomes of the benchmarking and optimization activities performed between M13 and M28. During the reporting period, the HPC infrastructure used by the project was extended to include newly added EuroHPC systems. The process of acquiring resources still remains cumbersome, and in addition the amount of awarded resources through the access schemes typically utilized by the pilots has been significantly reduced.

Despite the challenges faced with regard to resources, significant progress has been achieved in terms of performance optimisation, leading to achieving the targets set for 3 out of the 4 KPIs related to benchmarking and optimisation. More notably, one of the HiDALGO2 codes (UAP-Xyst) has already achieved scalability of up to almost 200k cores, running on the LUMI-C partition. On the other hand, more performance bottlenecks have been identified and all pilots are continuously working on improving their scalability. For the last part of the project, the pilots will focus on more profiling and bottleneck analysis to address communication and I/O challenges, improving deployment and portability via containerization, and scaling applications to extreme node counts.

References

[1] *Homepage of HiDALGO2 CoE*, https://www.hidalgo2.eu/ [retrieved: 2025-04-24]

[2] K. Nikas, et al. *HiDALGO2 D3.1 - Scalability, Optimization and Co-Design Activities*, 2023, http://dx.doi.org/10.13140/RG.2.2.16877.15849

[3] *Score-P wiki page*, https://hpc-wiki.info/hpc/Score-P [retrieved:2025-04-24]

[4] *Homepage of CUBE,* https://www.vi-hps.org/Tools/Cube.html [retrieved:2025-04-24]

[5] *Homepage of EuroHPC JU*, https://eurohpc-ju.europa.eu [retrieved: 2025-04-24]

[6] *Homepage of PSNC*, https://www.psnc.pl/ [retrieved: 2025-04-24]

[7] *Homepage of OpenFOAM*, https://www.openfoam.com/ [retrieved: 2025-04-24]

[8] L. Környei, Z. Horváth, A. Ruopp, A. Kovacs, and B. Liszkai. “Multi-scale Modelling of Urban Air Pollution with Coupled Weather Forecast and Traffic Simulation on HPC Architecture”. In Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Companion (HPC Asia 2021), 2021

[9] J. Bakosi, “Open-source complex-geometry 3D fluid dynamics for applications with unpredictable heterogeneous dynamic high-performance-computing loads”, Computer Methods in Applied Mechanics and Engineering, Volume 418, Part B, 2024.

[10] RedSIM documentation, https://redsim.mathso.sze.hu/about/overview [retrieved: 2025-04-28]

[11] *SimpleFoam documentation,* https://doc.openfoam.com/2212/tools/processing/solvers/rtm/incompressible/simpleFoam/ [retrieved: 2025-04-24]

[12] *PimpleFoam documentation*, https://doc.openfoam.com/2212/tools/processing/solvers/rtm/incompressible/pimpleFoam/ [retrieved: 2025-04-24]

[13] *Github page of Zoltan*, https://sandialabs.github.io/Zoltan [retrieved: 2025-04-24] (2024)

[14] *Ktirio Urban Building Framework*, https://feelpp.github.io/ktirio-urban-building/ktirio-urban-building/index.html[retrieved: 2025-04-24]

[15] *IFC: Industry Foundation Classes standards*, https://technical.buildingsmart.org/standards/ifc/ [retrieved: 2025-04-24]

[16] *OpenStreetMap Wiki*, https://wiki.openstreetmap.org/wiki/Main_Page [retrieved: 2025-04-24]

[17] *Homepage of Modelica association*, https://modelica.org/ [retrieved: 2025-04-24]

[18] *Homepage of FMI standard*, https://fmi-standard.org/ [retrieved: 2025-04-24]

[19] *Github page of feel++*, https://github.com/feelpp/feelpp[retrieved: 2025-04-24]

[20] *Homepage of Docker*, https://www.docker.com [retrieved: 2025-04-24]

[21] *Homepage of Apptainer*, https://apptainer.org/ [retrieved: 2025-04-24]

[22] *WRF wiki page*, https://en.wikipedia.org/wiki/Weather_Research_and_Forecasting_Model [retrieved: 2025-04-24]

[23] *Open wildland fire modelling Wiki,* https://wiki.openwfm.org/wiki/WRF-SFIRE [retrieved: 2025-04-24]

[24] *Homepage of EPICURE*, https://epicure-hpc.eu [retrieved: 2025-04-24]

[25] H. Hersbach et al., "The ERA5 global reanalysis". Quarterly Journal of the Royal Meteorological Society, 2020.

[26] P.L. Andrews, "Behaveplus fire modeling system: past, present, and future". In Proceedings of 7th symposium on fire and forest meteorology, pages 23-25. American Meteorological Society Boston MA, 2007.

[27] WPS V4 geographical static data downloads page. https://www2.mmm.ucar.edu/wrf/users/wrf_users_guide/build/html/running_wrf.html [retrieved: 2025-04-25] NCAR (2024).

[28] Vampir wiki page, https://hpc-wiki.info/hpc/Vampir [retrieved: 2025-04-24]

[29] Chair for System Simulation, *"waLBerla (widely applicable Lattice Boltzmann from Erlangen)". Zenodo, Oct. 30, 2023.*

[30] M. Bauer et al., "waLBerla: A block-structured high-performance framework for multiphysics simulations". Computers & Mathematics with Applications, 2020.

[31] *Homepage of ReFrame*, https://reframe-hpc.readthedocs.io/en/stable/ [retrieved: 2025-04-24]

[32] M. Holzer, "Code generation in a lattice Boltzmann framework for exascale computing". Dissertation published at openfau, 2025.

[33] M. Holzer et al, "Highly efficient lattice Boltzmann multiphase simulations of immiscible fluids at high-density ratios on CPUs and GPUs through code generation". The International Journal of High Performance Computing Applications, 2021.

[34] Chair for System Simulation (Friedrich-Alexander Universität Erlangen-Nürnberg), "lbmpy". Zenodo, Oct. 28, 2024.

[35] M. Bauer et al, "lbmpy: Automatic code generation for efficient parallel lattice Boltzmann methods". Journal of Computational Science, 2021.

[36] F. Hennig et al, "Advanced Automatic Code Generation for Multiple Relaxation-Time Lattice Boltzmann Methods". SIAM Journal on Scientific Computing, 2023.

[37] Chair for System Simulation (Friedrich-Alexander Universität Erlangen-Nürnberg), "pystencils". Zenodo, Oct. 28, 2024.

[38] C. Alt, M. Lanser, J. Plewinski, A. Janki, A. Klawonn, H. Köstler, U. Rüde et al. "A continuous benchmarking infrastructure for high-performance computing applications". International Journal of Parallel, Emergent and Distributed Systems 39(4), 501–523, 2024.

[39] F. Galeazzo, et al. *HiDALGO2 D3.7 - Ensemble Scenarios for Global Challenges*, 2024, http://dx.doi.org/10.13140/RG.2.2.28421.15841

[40] Online Xyst report, "https://www.hidalgo2.eu/scaling-to-new-heights-xyst-code-excites-with-unprecedented-196k-core-run-on-lumi-supercomputer", [retrieved: 2025-04-29]