D3.4 Third update on the assessed applications/codes
Version 1.0

Document Information

Contract Number	101143931
Project Website	www.pop-coe.eu
Contractual Deadline	M18, June 2025
Dissemination Level	PU
Nature	R
Author	Christian Terboven (RWTH)
Contributor(s)	Radim Vavřík (IT4I@VSB)
Reviewer	Ondřej Vysocký (IT4I@VSB)
Keywords	Performance optimization, Scalability, Efficiency

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101143931. The JU receives support from the European Union's Horizon Europe research and innovation programme and Spain, Germany, France, Portugal, and the Czech Republic.

Change Log

Version	Author	Description of Change
V0.1	Christian Terboven	Initial Draft
V0.2	Christian Terboven	Chapters 1, 2 and 4 complete, about half-way done with Chapter 3 (for internal review)
V0.3	Christian Terboven	All chapters completed (for internal review)
V0.5	Christian Terboven	Response to internal review
V1.0	Elena Markocic	Document formatted for submission

Table of Contents

Executive Summary

This deliverable includes a summary of the work carried out by Task 3.1 Assessments during the months 12 to 18 of the project.

1. Introduction

In this document, we report the data gathered and generated from the performance assessments requests[1] received by the project. The assessments handled by the project go through several states. When a request for (assessment) service is received, its status is *Received*, once it has been accepted and assigned to one of the partners, we consider it *Open.* An open assessment can be *Blocked* for technical, legal, or other reasons. When an assessment is closed successfully after presenting the results to the customer, we consider it *Completed*. An *Open* or *Blocked* assessment can be cancelled if unsolvable issues are detected. Further process details including our data collection methodology have been presented in Deliverable D3.2.

The document is divided into three main parts. The first one (Chapter 2) characterizes up to 60 requests and codes received, which means it considers all the requests of assessments *Received*, *Completed,* and *Open*, where data was available at the end of May 2025. The data used in this part of the report comes from the information provided by the customer in the two initial forms[2], partially annotated or extended by the POP3 performance analysts while carrying out the assessment. This data includes the parallel programming models supported, the HPC cluster requested for the assessment, or the source of the request, among others. This information is relevant to the Tools (T4.3) and Methodology (T4.4) tasks as it guides the main efforts in the development of new features of the tools, which use case the methodology should consider, or which architectures must be supported.

The second part (Chapter 3) of this document analyses the performance data gathered from 10 *Completed* assessments since M12. This section of the document is based on the information provided by the analysts in the final assessment form and the final report.

Finally, the third part (Chapter 4) of this deliverable briefly summarizes the *Open* assessments.

[1] https://www.pop-coe.eu/services
[2] https://pop-coe.eu/request-service-form

2. Assessments characterization

In this section, we analyse the characteristics of the requests received by Task 3.1 Assessments. The data presented here was assessed by the end of May 2025.

2.1 Origin of assessments

One of the main focuses of POP3 is to analyse the codes of the coexisting EuroHPC Centres of Excellence (CoEs). Currently, we have received service requests from 8 different CoEs: CEEC, ChEESE2, EoCoE3, ESiWACE, EXCELLERAT, MultiXscale, Plasma-PEPSC and SPACE.

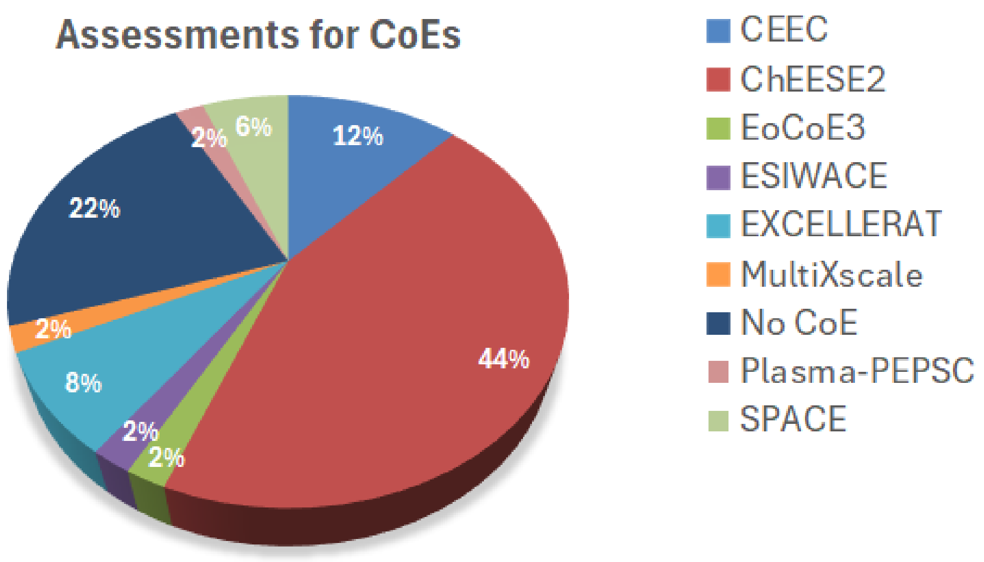

Figure 1: Distribution of source of performance assessments by CoE

Figure 1 shows the distribution of requests received by the different CoEs and from other customers (category: No CoE).

2.2 Cluster used for assessment

When a request for a performance assessment is received, the customer decides in which cluster the code should be assessed. Figure 2 shows the distribution of the clusters used for the performance assessments. We can observe a high heterogeneity in the cluster requested for the analysis, which must be reflected in WP4 (Tools) to focus on supporting a large variety of hardware. The list of clusters includes the European pre-exascale supercomputers.

With 14 assessments, Leonardo is the most popular cluster, which was the preferred system, especially in the ChEESE2 campaign. Further, LUMI and MN5 are very popular with 11 assessments. Just considering these three systems, it can be noted that they exhibit different architecture in terms of CPU (Intel vs. AMD) and GPU (NVIDIA vs. AMD). In particular, the differences in these two HPC accelerator technologies and lower maturity of the AMD software stack underlines the technical challenges solved in the respective assessments.

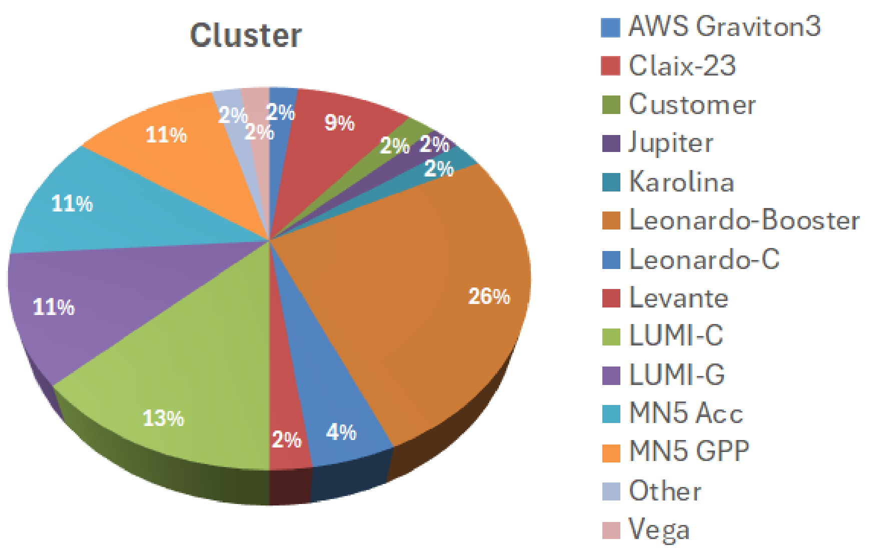

Figure 2: Distribution of clusters used for performance assessments

2.3 Programming models

In this subsection, we analyse the (parallel) programming models used in the codes assessed. Regarding parallelism, we analyse three different levels: the distributed programming model (MPI), shared memory programming model (threading), and the use of accelerators (GPUs).

First, Figure 3 shows the use of programming languages in the assessments. C++ and the mixture of C and C++ clearly dominate. Fortran has a fraction of 38%. A small portion of codes use Python in combination with C++.

Figure 4 shows the use of MPI: 94% of the codes analysed use MPI for distributed memory parallelism. The rest is shared memory and/or GPU only.

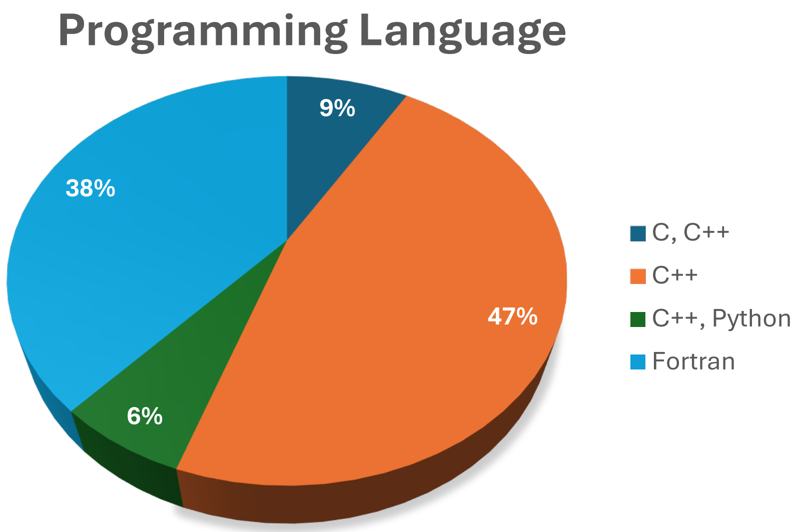

Figure 3: Use of programming languages

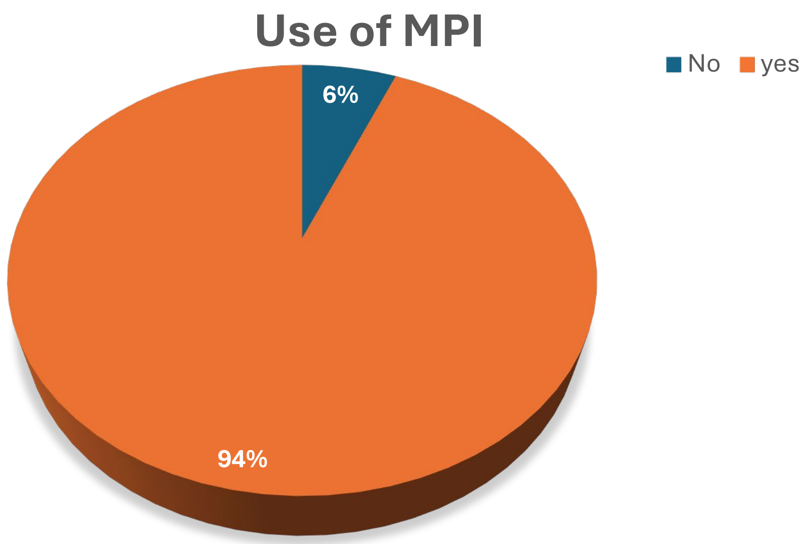

Figure 4: Use of MPI / distributed memory parallelism

Figure 5 shows the distribution of programming models used at the threading level for shared memory parallelism, which is relevant for 37% of the codes. Out of these, OpenMP continues to be the most prominent model. But as we also encountered other combinations of threading programming models, which again poses a challenge for the tools and methodology tasks, as analysts must be able to support all of them and - in particular - their combination.

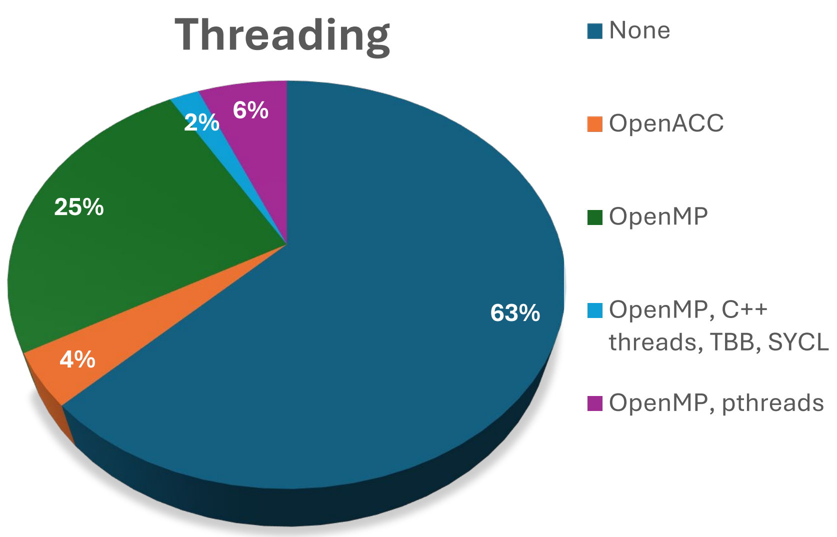

Figure 5: Use of threading models for shared memory parallelism

With 36% significantly less than half of the codes did not use GPUs at all. The most-used GPU programming models are CUDA and OpenACC, as shown in Figure 6.

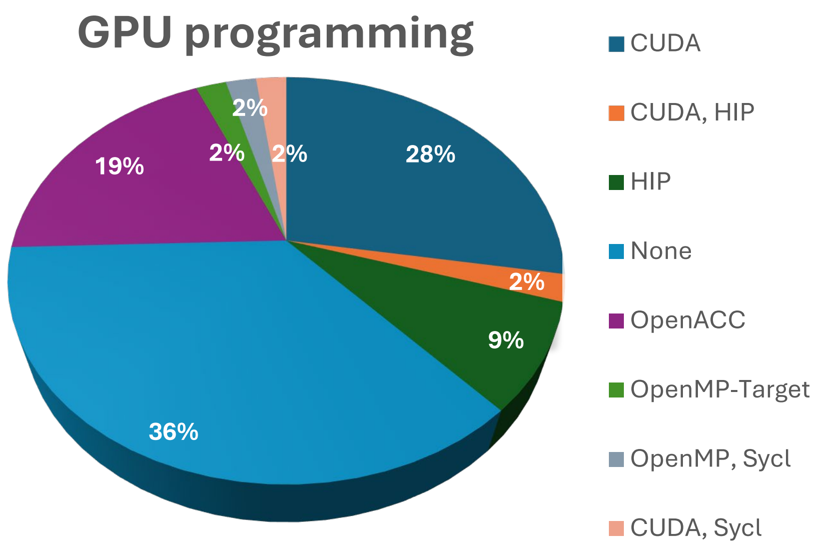

Figure 6: Use of GPU programming models

In this case, the feedback for the Tools task in WP4 is to prioritize support for CUDA and OpenACC. While, if possible, provide support for OpenMP and SYCL.

3. Completed assessments summary

In this section, we analyse the data gathered from the *Completed* assessments since M12. The data presented here was assessed by mid of June 2025.

3.1 Scaling of codes assessed

Figure 7 shows the maximum number of CPU cores used in the *Completed* assessments and the number of assessments using a certain number of resources. We see a broad distribution in the number of cores used. The highest values being 12 288 cores.

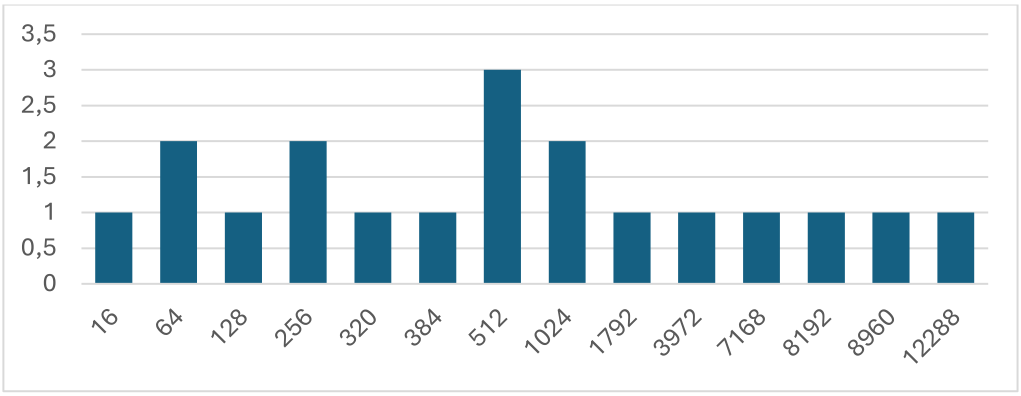

Figure 7: Maximum number of cores used per assessment

The total number of GPUs used in assessments ranges from 16 to 512 as shown in Figure 8.

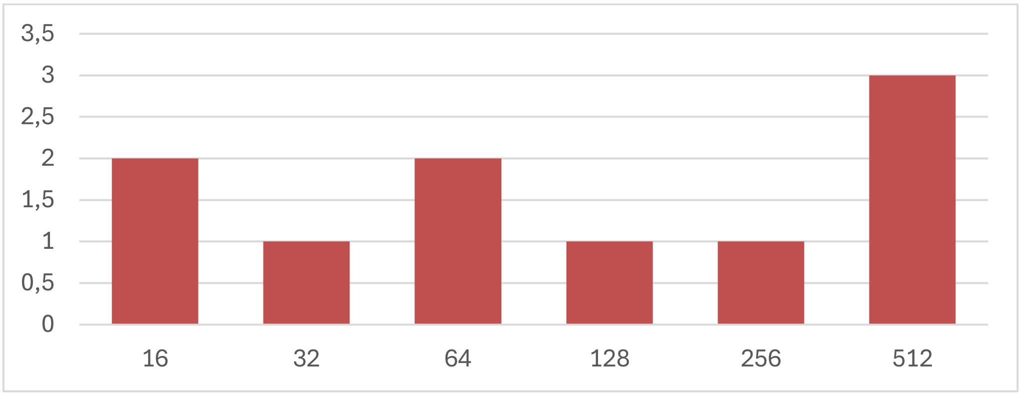

Figure 8: Maximum number of GPUs used per assessment

3.2 Analysis of Efficiencies measured

In this section, we briefly analyse the performance gathered from the performance assessments that have been *Completed*. The extended number of metrics gathered as part of the POP methodology has been described in D3.2.

To avoid having data that is too sparse, we have continued to compute statistics only for some high-level metrics. We have categorized the efficiencies as follows:

- **Super**: Higher than 100%
- **Very good**: Between 100% and 90%
- **Good**: Between 90% and 80%
- **Fair**: Between 80% and 70%
- **Poor**: Between 70% and 60%
- **Bad**: Lower than 60%

In Figure 9, we plot for each metric how many assessments received each kind of categorization.

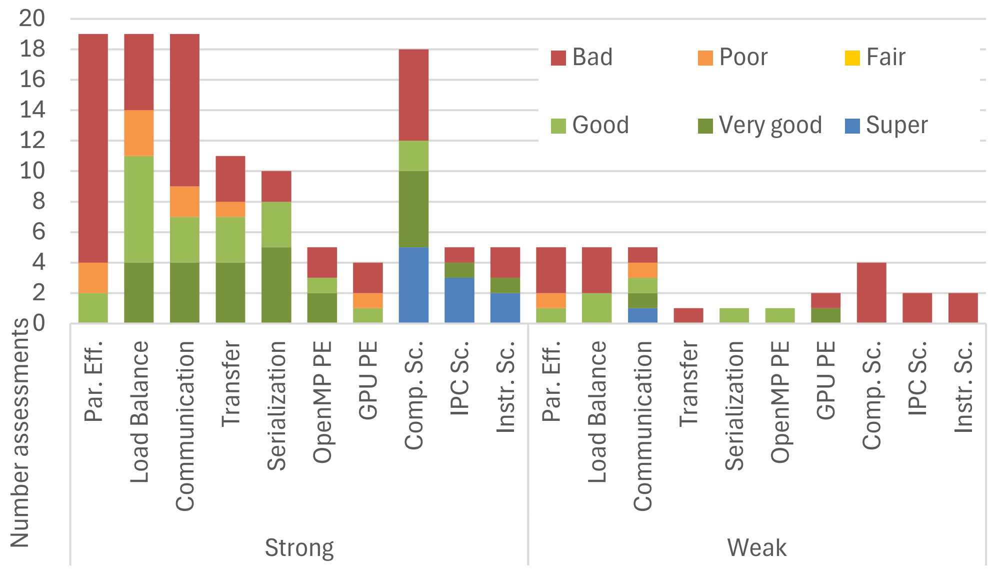

Figure 9. Categorization of the efficiencies measured for the Completed performance assessments

The distribution did not change significantly from the previous six months that were reported in D3.2 in that we observe a high variation. This underlines, first, the value of the POP metrics in providing a holistic performance characteristic supporting a wide range of applications, and second, the need for continued performance assessments of codes running on modern (large-scale) HPC systems.

The following subsections detail the assessments that were completed during the reporting period.

3.3 POP3_AR_016

The analyzed application is a **mini-application extracted from an IFS model.** It performs a **global spherical harmonics transformation.** The performance assessment identified a **major issue: poor orchestration efficiencies** related to **MPI communication and CUDA management**. A key finding is that **most of the runtime is spent on frequent MPI communication and CPU↔GPU data transfers**, while **useful computation forms only a small part** of the overall execution time.

The **initialization phase accounts for 52% of the total runtime**, with processes 0 and 15 spending significant time in `MPI_Allgatherv`, primarily involving CPUs. The initial step of the iterative computation takes 22% of the runtime, characterized by GPU↔CPU data transfers, CUDA data loadings, and device memory allocations, leading to imbalances and longer `MPI_Barriers` for certain ranks. The **Region of Interest (RoI), where GPUs are mainly used, only sees GPUs actively computing for approximately 2 seconds, which is just 17% of the RoI's runtime**; the rest of the time is spent idling, in data transfers, and synchronization routines. Further, the application suffers from **frequent waiting in blocking collective communication, barriers, and wait-all calls,** with **54% of the time spent on data transfers (CPU-GPU and MPI) on average.** The **MPI synchronization part, particularly barriers, significantly expands with more resources**, growing from 9% to 29% of the total runtime. While strong scaling is good for 4 and 8 compute nodes, it begins to stagnate at 16 nodes and does not scale well at 32 nodes. Furthermore, there is an **increasing load imbalance with more resources.**

To improve the scalability of the application, several recommendations were made. It is suggested to **review the current communication pattern**, assessing whether all communication and data transfers are necessary and if the volume of data sent in every step is genuinely required. There's also a recommendation to **explore overlapping communication with computation.** Furthermore, **focus should be placed on achieving a more balanced workload distribution** across GPUs. Finally, it is advised to **reappraise the necessity of short kernels and their regular synchronization with the host** to mitigate overheads from stream synchronization and kernel launches.

3.4 POP3_AR_017

The application provides a numerical solution for equations governing fluid flow in three dimensions, specifically used to **simulate the impact of airplane wing shape on airflow.** The key finding of the analysis was that **scaling is good up to 128 GPUs**, but beyond this point, **communication begins to present a significant problem** for the specific input problem size and parametrization tackled. The code uses OpenACC as the GPU programming model.

It was profiled scaling from 16 MPI ranks and 16 GPUs up to 512 MPI ranks and 512 GPUs. While using more GPUs generally resulted in faster execution of time-steps and reduced time between MPI calls, **Parallel Efficiency was observed to be ≥ 90% only up to 64 GPUs.** Beyond this, Communication Efficiency, primarily due to Transfer Efficiency, significantly decreased, especially with 128 or more GPUs. Furthermore, analysis revealed that **host-device transfers account for a significant portion of execution time** and **do not overlap with GPU kernels. Nvidia N**sight Compute analysis highlighted issues like **uncoalesced global and shared memory accesses, and occupancy bottlenecks** in several computationally intensive kernels.

To improve the scalability, several recommendations were identified. Firstly, addressing the **communication pattern** that causes contention in inter-node input links, potentially by **reordering inter-process communication**, could reduce communication costs and improve efficiency. Secondly, the **low parallel efficiency during communication phases on large runs could be improved by advancing or delaying computations** to better overlap communication and computation. Specifically, introducing **overlap for host-device transfers**, or even **reducing their size and frequency by minimizing dependencies and retaining data on GPUs for longer periods**, could substantially enhance performance. Lastly, significant improvements could be achieved by **optimizing GPU kernels** to mitigate **global memory coalescing issues** (with potential gains of 41-66%), addressing **uncoalesced shared memory accesses** (35-39% gain), and **improving occupancy** (up to 40% speedup potential on some kernels).

3.5 POP3_AR_018

This is a **Computational Fluid Dynamics (CFD)** code, primarily written in **Fortran** and utilizing **MPI** for parallelization. The simulation was executed with 5, 10, 20, 40, and 80 nodes. The key finding of the analysis was that the **primary issue impacting performance was Serialization, largely due to the coupling process** where one physics waits while the other is running. Additionally, **common issues across the solvers included load imbalance caused by long computational bursts with low Instructions Per Cycle (IPC) due to cache misses, and system noise affecting serialization.**

Regarding its performance characteristics, the application's **total execution time began to separate from the ideal time at 40 nodes** when analyzing strong scalability over 1000 time steps. Across the individual solvers, various efficiency issues were observed: **ALEFOR experienced load balance drops (from 90% to 74% at 80 nodes), serialization, and transfer efficiency decreases, with load imbalance linked to IPC and instruction imbalances. NASTIN showed similar drops in load balance and issues with serialization and transfer efficiency, with load imbalance in its "Element Assembly" part correlated with high L1 cache misses and in its "Solver Continuity" part with higher L3 cache misses. SOLIDZ exhibited low load balance and serialization at 80 nodes, primarily due to long bursts with low IPC caused by cache misses.**

One specific suggestion was made related to the identified "noise" affecting serialization in ALEFOR was made: **this kind of noise can be avoided by reducing the number of MPI collective calls**, as other ranks wait for the affected one

3.6 POP3_AR_019

The analyzed application is an open-source version of a commercial code primarily used for **crash simulation and structural mechanics.** It is developed using Fortran, OpenMP, and MPI programming languages. The key finding of the analysis indicates that **OpenMP parallelism has limitations**, largely due to critical sections and locks within the code, suggesting that it should ideally be limited to no more than 4 OpenMP threads per MPI rank. In stark contrast, **MPI parallelism, both within and between nodes, achieves remarkable speed-up with limited time spent in MPI activities and very good load distribution.**

The application targets "single" runs on a few thousand cores, though typical users often launch multiple parallel runs with slightly varied configurations. It employs an **explicit method with very small time steps**, making **numerical accuracy and reproducibility major concerns.** Consequently, any code transformations that might impact numerical accuracy, such as fast math compiler flags or fused multiply-add (FMA) options, are strictly prohibited. Already very long parallel run times imply that a single-core run would take several days, rendering classical speed-up measurements impractical and leading to huge trace files if full tracing were performed. Specific issues identified include **significant inactive time and poor resource usage in OpenMP configurations**, primarily attributed to critical sections and locks.

To improve the scalability, the analysis offers several recommendations. It is advised that **OpenMP parallelism should preferably be limited to not more than 4 OpenMP threads per MPI rank.** Furthermore, it is critical to **investigate limiting the impact of critical sections** in OpenMP, even if suppressing them entirely proves difficult. Future steps outlined for performance optimization include further addressing **OpenMP limitations**, conducting **vectorization analysis and optimization**, performing **compiler analysis** (examining flags and comparing GNU Fortran with ACFL), and undertaking an **analysis of x86 versus ARM** architectures as a second level service.

3.7 POP3_AR_020

The performance assessment analysed a C/C++ application that utilizes **MPI and OpenMP** programming models. A key finding of the analysis was that **MPI Send-Receives and Group communication begin to dominate as the scale increases**, with the **largest run on 16 nodes showing degradation primarily in MPI serialization and OpenMP communication, both operating at 80% efficiency.**

The application's performance is characterized by a solver that includes four groups of bursts, with most of the execution time spent in very long bursts lasting approximately 1 to 30 seconds. While load balance efficiencies are generally high due to imbalances compensating over time across all steps, a significant portion of MPI time is consumed by collective operations such as `MPI_Allreduce`, `MPI_Alltoall`, `MPI_Sendrecv`, and `MPI_Allgather`. The MPI Serialization Efficiency (MPI SE) varies among routines, with DD1 showing 100% efficiency and GRAV, the longest routine, at 92%. However, DENS, a long routine with 86% MPI SE, is identified as a prime candidate for improvement.

To improve the scalability of the application, several potential steps are recommended. These include **implementing communication overlap with computation by using non-blocking communication.** Additionally, **optimizing collective operations** is advised to mitigate the increasing overhead of MPI communication at higher scales. Finally, **fusing OpenMP loops** is suggested as a way to enhance performance, particularly to address the observed small granularity in some OpenMP functions.

3.8 POP3_AR_021

The application analysed in the performance assessment is a code primarily used for **near-Earth plasma simulations.** The **key findings** highlight significant performance bottlenecks, primarily stemming from **instruction load imbalance** and **unneeded forced serialization of communication.**

The performance characteristics within the 'Propagate' slice (which constitutes 69% of the focus of analysis), reveal several issues. In one region, egion, there's an instantiation of **non-blocking sends and receives**, but communication is **forced to complete in order of instantiation**, involving many `MPI_Waitall` calls. There is significant data transfer, with total bytes transferred reaching 6.35TB for `transfer-stencil-data-z` and 2.6TB for `update_remote-z`. A major concern is **uneven bandwidth utilization**, which leads to **load imbalance due to communication**, and performance is more sensitive to bandwidth than latency. Furthermore, the `compute-mapping-z` sub-region exhibits an **instruction load imbalance** caused by uneven instruction distribution across MPI processes. In `Propagate-magnetic-field`, **OpenMP worksharing worsens the instruction load imbalance**, with one thread appearing to be the root cause of de-synchronization.

To improve the scalability of the application, the identified issues directly suggest areas for optimization. It is crucial to address the **unneeded forced serialization of communication** to allow for more asynchronous progression. Efforts should also focus on strategies to improve **uneven bandwidth utilization** and reduce the resulting load imbalance during communication phases.

3.9 POP3_AR_022

The assessed application is an OpenMP application. Written in **Fortran with OpenMP**, it incorporates LAPACK and BLAS routines and is designed to execute 192 energy steps using strong scaling. This performance assessment aimed to re-evaluate a new version of the code after a previous POP2 assessment identified a routine limiting scalability. The **key finding** of this assessment is that, despite improvements in vectorization and a shifted hotspot, the application **does not scale even to a low number of threads**, exhibiting a **significant speedup decrease from 48 to 96 cores.** Currently, **90% of the total runtime is concentrated in the LAPACK routine zgesv**, which has become the primary bottleneck for scalability.

The application's performance characteristics reveal several issues contributing to its poor scalability. On the CPU side, a significant problem is that **clock frequency scales down when all cores of a socket utilize AVX512 instructions**, impacting performance as core count increases. Memory-related issues are also prominent, with an **increasing number of remote NUMA accesses** observed as thread count rises, reaching 68.1% with 96 threads from 0.0% with 12 threads. Additionally, **OpenMP imbalances** were identified in certain loops, where inner-loop iterations decrease significantly for outer-loop iterations scheduled to high thread IDs.

To improve the application's scalability, several recommendations have been put forth. To address the issue of **increasing remote NUMA accesses**, it is suggested to **reduce them by using MPI and allocating chunks per process**. To counter the **frequency decrease and CPU throttling** experienced when many cores use vector instructions, a primary recommendation is to **offload the zgesv kernel**, and potentially other parallel regions, to a **GPU**, possibly utilizing GPU-enabled libraries like MAGMA and OpenMP Target Offloading. For the identified **OpenMP imbalances** in specific parallel loops, switching to a **dynamic schedule** from the implicit static one is advised. Finally, further **OpenMP parallelization** should be implemented.

3.10 POP3_AR_023

The application is an **N-Body Simulation**. It was assessed on a single compute node, utilizing 2x Intel Sapphire Rapids CPUs and 4x NVIDIA H100 GPUs, as the application uses OpenMP and OpenMP-Target (GPU) programming models and was tested with input cases ranging from approximately 100,000 to 1 million bodies. The **key finding** of the analysis was that **full resource utilization (of 4 GPUs) was not achieved, even with the largest problem size**, and for smaller problem sizes, there was **no scaling beyond 2 GPUs.**

The performance characteristics indicate several areas of concern. The application has a single-threaded initialization phase, while the integration phase involves one OpenMP thread per GPU, with iterative calls of a single kernel and small data transfers and I/O between iterations. While there were no major issues in work distribution across GPUs, showing almost perfect load balancing and no significant efficiency decrease due to data transfers, the CPU remained idle during GPU computation, leading to wasted CPU resources. The analysis highlighted **low memory throughput**, which further decreased with the use of more GPUs. This low memory utilization was attributed to **low GPU occupancy**, which was also found to be decreasing with more GPUs. The theoretical occupancy was noted to be low at 43.75% due to too many registers being used per thread, and the achieved occupancy was even lower due to a partial wave (tail loop). Other observations included minor load balance issues on the GPU, likely due to the partial wave, and an incorrect usage of the OpenMP standard where device pointers were not marked correctly.

To improve the scalability of the application, several recommendations were put forward. It was suggested that **more fine-grained loop distribution** could increase achieved occupancy. Specifically, parallelizing the inner loop (reduction) and distributing the outer loop could lead to more small kernels and higher occupancy, potentially also reducing registers per thread. Crucially, the **device pointers must be correctly marked with the `is_device_ptr` clause** in the OpenMP target pragma. To mitigate the tail effect, increasing the number of blocks launched was recommended. Furthermore, addressing the underutilized CPU was suggested, possibly by partitioning bodies for CPU processing as well, complementing the GPU partitioning. The analysis also noted that OpenMP Target provides limited room for optimization, implying that **other compilers might offer better performance or more flexibility** (NVHPC 24.9 was used).

3.11 POP3_AR_024

The performance assessment was on a software package designed for the **simulation of seismic wave propagation** based on the spectral-element method. The assessment was carried out on the **Leonardo-Booster** HPC system. This application, which uses Fortran90 (with some C) and is parallelized with MPI and CUDA (one MPI process per GPU), was assessed the ChEESE CoE. The key finding of the assessment, particularly regarding the 'v2' revised version, was a **considerable performance improvement for the largest execution configuration involving 512 GPUs**, where its strong scaling efficiency increased significantly from 51% to **67%.** This improvement was primarily attributed to **much more efficient orchestration of GPU devices via the exploitation of GPU-Aware MPI.**

The code exhibits **excellent weak scaling**, which is expected to continue to higher node counts, and showed no change in this characteristic with the v2 updates. For strong scaling, the application achieved **very good performance (above 80% of perfect)** up to approximately 64 compute nodes, which corresponds to 256 GPUs. The core of the analysis focused on the `xspecfem3D/iterate_time` solver routine, which includes all CUDA kernel executions. In the initial assessment, performance limitations at higher GPU counts (256 and 512 GPUs) included growing GPU idle time and substantial

CPU computation time due to explicit host-device data transfers, with MPI communication no longer fully overlapping GPU kernel computation at 512 GPUs. With the 'v2' revisions, while GPU computation time still slowly grew, **CPU computation time became negligible** as the explicit data transfers were eliminated. Furthermore, CUDA kernel execution was approximately **1.5% faster** on 512 GPUs, and overall GPU idle time was reduced.

The significant improvements in strong scaling, particularly for larger configurations, in the 'v2' assessment highlight specific recommendations for enhancing the scalability of HPC applications like this one. The primary recommendation is to **migrate from non-GPU-Aware MPI libraries (such as Intel oneAPI MPI) to GPU-Aware MPI implementations.** This transition **eliminates the need for explicit data transfers between the host and device**, thereby making CPU computation time negligible and improving orchestration efficiency. Additionally, it was suggested that using **optimized and current compiler and runtime environments** can yield further performance gains

3.12 POP3_AR_025

The analyzed application is a high-performance computing application for **Nuclear Fuel Cycle Simulation**, encompassing the entire cycle from fuel production and consumption in reactors to cooling, reprocessing, and waste storage. It is primarily written in **C++ and OpenMP** (without MPI).

The analysis identified that the code's **single-core performance is limited by poor vectorization and significant time spent in the ROOT library**, with only 45% of execution time in user code. The OpenMP version shows **only minor gains up to 4 cores before experiencing a slowdown**, largely due to **active waiting (`OMP_Wait`).** In contrast, the **concurrent weak scaling mode exhibited very good performance**, with only about a 10% slowdown when scaling from 1 to 52 instances, indicating efficient handling of parametric exploration workloads.

To improve the scalability of the application, several recommendations were put forth. For the single-core version, since only a small fraction of the large General Purpose ROOT library is utilized, it might be beneficial to **replace it with a custom implementation of the specifically needed routines.** Furthermore, the code is currently not vectorizable, meaning **major refactoring would be required to enable vectorization** and achieve potential gains in this area. For the OpenMP version, the analysis concluded that **simple parallelization at the innermost loop level does not yield significant benefits** beyond 4 cores. Given the nature of the code, which consists mostly of independent components, **Task Parallelism could be a more effective approach** to improve scalability, though this would also necessitate a major refactoring effort.

4. Open assessments summary

At the time of this writing, POP3 has 34 *Open* assessments. 10 of theses were not requested by any EuroHPC Centre of Excellence, 1 by SPACE, 1 by ESIWACE, 1 by EoCoE3, 2 by Plasma-PEPSC, 3 by EXCELLERAT, 5 by CEEC, 12 by ChEESE2.

Of the *Open* assessments, there are 4 that do not use MPI, but only OpenMP. There are 17 performance assessments that require the use of GPUs.

All of these assessments are ongoing, except for one that is blocked due to an NDA requirement.

5. Conclusion

During the M13-M18 months of the project reported here, we have finalized 10 performance assessments. One additional assessment is expected to be closed by the end of June and will be reported in the next deliverable D3.5 [M24].

Most of these assessments have been done for other EuroHPC Centres of Excellence. These assessments have been challenging because they use state-of-the-art features of the programming models. When aspects were not fully / well supported by some of our tools, this information was passed on to the tool improvement task in WP4. In these cases and when necessary, we have used alternative tools or methods to be able to conduct the assessments.

One other notable challenge continues to be to get the codes and the inputs from the CoE customers in a timely manner.

Of all the service requests received, we see that most of them continue to require one of the European pre-exascale systems and our tool development is planned accordingly.

Regarding the programming models used, we observe that the standard de-facto continues to be MPI for the distributed memory level, while for the threading and GPU levels, there is more variety, the most used ones: OpenMP, CUDA, and OpenACC.