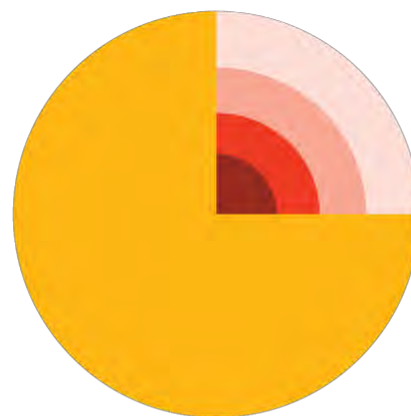




ChEESE

Deliverable D2.2 First report on improving code performance at exascale



Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Spain, Italy, Iceland, Germany, Norway, France, Finland and Croatia under grant agreement No 101093038.



ChEESE-2P Deliverable

D2.2 First report on improving code performance at exascale

Document Information

Contract Number	101093038
Project Website	https://cheese2.eu
Contractual Deadline	31/12/2024 (M24)
Activity	WP2
Dissemination Level	PUBLIC
Nature	R

1. Copyright

Copyright Members of ChEESE collaboration, 2023/2026.

2. Delivery Slip

Role	Name	Partner/Activity
Author	Piero Lanucara	CINECA
Contributor 1	Giorgio Amati	CINECA
Contributor 2	CINECA+code owners	All

Reviewer 1	Martin Rose	HLRS
Reviewer 2	Okba Hamitou	Eviden

3. Version control

Version	Date	Description of change	Author/Partner
V1	21/11/2024	Initial draft	PL
V2	06/12/2024	Final draft	PL/GA
V3	20/12/2024	Final	PL

4. List of Acronyms

Acronym	Description
AMR	Adaptive Mesh Refinement
CAD	Code-Audit Driven
FOM	Figure of Merit
HBM	High Bandwidth Memory
HPC	High Performance Computing
MPI	Message Passage Interface
PD	Pilot Demonstrator
PETSc	Portable, Extensible Toolkit for Scientific Computation
PP	Performance Portability
PPC	Performance Portability Campaign
PSHA	Probabilistic Seismic Hazard Assessment
SC	Scientific Challenge

SDV	Software Development Vehicle
UQ	Uncertainty Quantification

INDEX

1. Executive Summary	7
2. Ambitions.....	9
2.1 Increasing the PP of flagship codes for different accelerators (T2.2)	10
2.2 Optimisation for exascale (T2.3)	11
2.3 Continuous Performance Efficiency Monitoring (T2.4)	11
3. Methodology.....	12
3.1 T2.2 Methodology.....	12
3.1.1 Porting.....	12
3.1.2 Targeting EuroHPC and SDV accelerated systems.....	13
3.1.3 Targeting mini apps based on increased portability status	14
3.1.4 Performance Portability Campaign	14
3.1.5 T2.3 Methodology.....	15
3.2.1 Targeting EuroHPC general purpose systems.....	16
3.2.2 Code Audit driven improvements.....	16
3.2.3 Other improvements.....	18
3.3 Methodology T2.4	19
3.3.1 TALP enhancement towards Exascale.....	19
3.3.2 TALP performance monitoring.....	20
4 Flagship codes	25
4.1 SeisSol.....	25
4.1.1 Flagship code/Mini-app(s) initial status.....	26
4.1.2 Porting.....	27
4.1.3 Performance Portability campaign	28
4.1.4 Code Audit driven improvements.....	31
4.1.5 Other improvements	31
4.1.6 Exascale ambition.....	31
4.2 SPECFEM3D	32
4.2.1 Flagship code/Mini-app(s) initial status.....	32
4.2.2 Porting	33
4.2.3 Performance Portability campaign.....	33
4.2.4 Code Audit driven improvements	35

4.2.5 Other improvements.....	36
4.2.6 Exascale ambition.....	37
4.3 ExaHyPE.....	37
4.3.1 Flagship code/Mini-app(s) initial status.....	37
4.3.2 Porting.....	38
4.3.3 Performance Portability campaign.....	39
4.3.4 Code Audit driven improvements.....	41
4.3.5 Other improvements.....	43
4.3.6 Exascale ambition.....	43
4.4 Tandem	44
4.4.1 Flagship code/Mini-app(s) initial status.....	44
4.4.2 Porting.....	44
4.4.3 Performance Portability campaign	47
4.4.4 Code Audit driven improvements.....	51
4.4.5 Other improvements	51
4.4.6 Exascale ambition	52
4.5 xSHELLS.....	52
4.6 HySEA.....	52
4.6.1 Flagship code/Mini-app(s) initial status.....	53
4.6.2 Porting.....	53
4.6.3 Performance Portability campaign	54
4.6.4 Code Audit driven improvements.....	60
4.6.5 Other improvements.....	60
4.6.6 Exascale ambition.....	60
4.7 FALL3D.....	60
4.7.1 Flagship code/Mini-app(s) initial status.....	61
4.7.2 Porting.....	63
4.7.3 Performance Portability campaign.....	63
4.7.4 Code Audit driven improvements.....	70
4.7.5 Other improvements.....	74
4.7.6 Exascale ambition.....	74
4.8 OpenPDAC.....	75
4.8.1 Flagship code/Mini-app(s) initial status.....	75
4.8.2 Porting.....	76
4.8.3 Performance Portability campaign	76

4.8.4 Code Audit driven improvements.....	76
4.8.5 Other improvements	77
4.8.6 Exascale ambition	81
4.9 LaMEM.....	83
4.9.1 Flagship code/Mini-app(s) initial status.....	83
4.9.2 Porting.....	83
4.9.3 Performance Portability campaign	84
4.9.4 Code Audit driven improvements.....	97
4.9.5 Other improvements.....	97
4.9.6 Exascale ambition.....	97
4.10 pTatin3D.....	98
4.10.1 Flagship code/Mini app(s) initial status.....	98
4.10.2 Porting.....	99
4.10.3 Performance Portability campaign	99
4.10.4 Code Audit driven improvements.....	100
4.10.5 Other improvements.....	100
4.10.6 Exascale ambition	100
4.11 ELMER/ICE	100
4.11.1 Flagship code/Mini-app(s) initial status	100
4.11.2 Porting	101
4.11.3 Performance Portability campaign.....	102
4.11.4 Code Audit driven improvements.....	108
4.11.5 Other improvements.....	108
4.11.6 Exascale ambition.....	108
5. Conclusions/Lessons learned.....	109
5.1 Porting	112
5.2 Performance portability campaign	112
5.3 Code audit driven improvements.....	113
5.4 Other improvements.....	113
5.5 Lessons learned.....	113
Appendix.....	116
List of the hardware used in the overall activities (main platforms).....	116

1. Executive Summary

D2.2 describes the methodology of WP2 aimed at achieving significant development in 10 out of 11 flagship codes of CHEESE-2P in terms of performance and performance portability. This methodology also seeks to identify a set of mini-apps and computational kernels that, together with the flagship codes, contribute to achieving these computational goals. The most noteworthy code-by-code results are as follows:

- **SeisSol:** Improved the features and performance of the current SYCL-based version. The performance portability campaign helped identify some intrinsic bottlenecks in the software stack on specific EuroHPC machines.
- **SPECFEM3D:** Enhanced the performance of the production-ready version in terms of large-scale communication efficiency and identified a highly useful benchmark to test the performance of various GPU programming models (CUDA and HIP).
- **ExaHyPE:** A GPU-enabled version developed using a portable language like OpenMP target offload was improved in multiple aspects (notably, memory layout and reduction of the number of loops in various kernels, task orchestration (via code audit), and fusion strategies). Additionally, a mini app was tested on different systems, revealing some performance bottlenecks on specific devices.
- **Tandem:** An initial GPU-enabled version was evaluated using suitable PETSc backends (particularly CUDA and HIP), making the code more portable across most of the accelerated EuroHPC architectures. This version was validated for numerical correctness and assessed for performance portability on NVIDIA and AMD devices. There is room for improvement, particularly on NVIDIA devices (issues with large memory usage in the CUDA backend).
- **HySEA:** The primary activity in D2.2 focused on the performance portability campaign of a version capable of exploiting CUDA and SYCL on suitable accelerated platforms (including Intel GPUs). Additionally, minor improvements to the production version were implemented, stemming from the code audit (CUDA kernels).
- **FALL3D:** A new, more performance-portable version of the code was developed. Moreover, a series of enhancements were introduced, including a drastic reduction in host-device memory transfers,

improvements to computational kernels performance and enhanced meteorological data handling.

- OpenPDAC: Improvements to the CPU-based code version were made, specifically in terms of numerical algorithm optimization and mesh quality enhancements, which halved the time-to-solution in large-scale runs.
- LaMEM: An extensive study was conducted on some of the computational kernels most responsible for LaMEM's performance on accelerated architectures (including NVIDIA GH200 and AMD). This was done using PETSc backends such as Kokkos, CUDA, and HIP.
- pTatin3D: Performance and preliminary performance portability of a key performance-relevant kernel for pTatin3D on accelerated architectures (primarily AMD) were assessed. Additionally, scalability improvements in parallelization were implemented and will be evaluated in the next code audit.
- ELMER / ICE: The ICE component of ELMER was ported to LUMI-G (a primary objective in WP2/D2.2) and its portability was evaluated using OpenMP target offload (matrix assembly) and suitable libraries for offloading linear algebra tasks on other accelerated architectures (notably AMGX on NVIDIA and rocALUTION on AMD).

2. Ambitions

The main objective of WP2 is to ensure that the evolution of the eleven flagship codes in ChEESE-2P will maintain the performance and portability expected by the project's computational and scientific challenges.

The strategy is built on top of the successful methodology already used in ChEESE original project (Grant agreement ID: 823844, <https://cheese-coe.eu/>) but extends its ambition by addressing the following new computational challenges in ChEESE-2P:

1. There are five new flagship codes in the project (Tandem, OpenPDAC, LaMEM, pTatin3D and ELMER/ICE).
2. The range of architectures available in the HPC market (particularly in EuroHPC) has now consolidated into an accelerated model (more than 75% of the peak in the Top500 list-see for example <https://www.nextplatform.com/2024/05/13/top500-supers-this-is-peak-nvidia-for-accelerated-supercomputers>).

The overall vision of WP2, and specifically the part related to tasks T2.2 and T2.3, is aimed at building an ecosystem of flagship applications and so-called mini apps (or mini-apps) that enable a guided evolution towards Exascale (in the context of the ChEESE project, the term mini app refers to simplified and standalone versions of more complex scientific applications).

The effectiveness and robustness of this approach is ensured on one hand by the interaction with WP3, which provides access to various mini apps (that can be either ready to use or quickly adapted to accelerated architectures), and on the other hand using precise computational metrics to drive the performance of the (future) "enhanced" application (see Fig. 2-1).

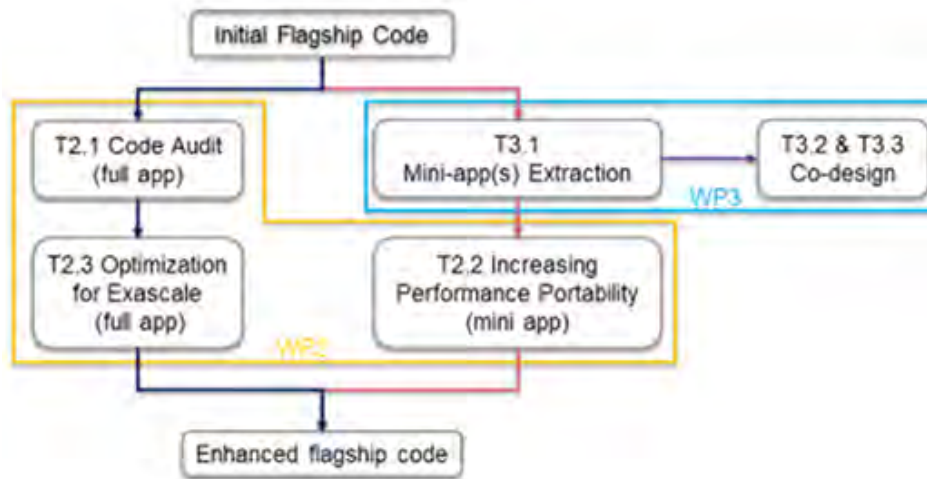


Fig. 2-1: the heaviest piece of the overall strategy in WP2 targeted to improve the performance and portability of flagship codes in ChEESE-2P. To be complemented with the performance efficiency monitoring (T2.4).

2.1 Increasing the PP of flagship codes for different accelerators (T2.2)

The objective of task T2.2 is to ensure that the ecosystem of flagship codes and mini apps meets the performance criteria as well as portability, particularly on accelerated architectures. The vision for the activities is aimed at addressing:

- A precise methodology (from literature) for evaluating performance portability, sufficiently robust and validated across different application fields and HPC architectures (see for example <https://doi.org/10.1109/MCSE.2021.3097276> and references therein)
- The use of software for performance portability that may include language intrinsic parallelism, directive-based programming models like OpenMP and OpenACC, performance portability framework (Kokkos), software layer for heterogeneous parallelism (like CUDA, HIP, SYCL and others) and GPU accelerated libraries. See <https://x-dev.pages.jsc.fz-juelich.de/models/> for a detailed list of available software including compatibility across different accelerators.
- Internal (to the project) dissemination of the tools addressed in the task.

This task does not depend on the code audit (task T2.1), but it is heavily reliant on the development of mini apps from task T3.1 of WP3. The architectures being exploited are the accelerated partitions of EuroHPC, but also SDVs that may become available from vendors or project partners to expand the range of exploration possibilities in T2.2.

2.2 Optimisation for exascale (T2.3)

Task T2.3 focuses on enhancing the overall performance and scalability of the flagship code towards Exascale. This task encompasses the entire “full” application (with certain specific exceptions), and the implementation strategy is based on the successful approach already developed and concretized in the WP2 of the first round of ChEESE. In particular, the following challenges are exploited:

- Single (heterogeneous) node performance
- Multi-node performance
- I/O improvement
- Algorithmic improvement
- Other aspects

The metrics used for the performance evaluation are the POP metrics (which were previously introduced in the deliverable D2.1 and will be employed for comparison in deliverable D2.3).

This task is heavily dependent on the code audit (task T2.1) and is naturally independent of WP3. All the available architectures (accelerated and non-accelerated, depending on the specific case) from EuroHPC can be used in this task.

2.3 Continuous Performance Efficiency Monitoring (T2.4)

The primary objective of task T2.4 is to ensure continuous performance monitoring throughout the entire project, not limited to the code audit procedure handled by task T2.1. The main tool is TALP (<https://doi.org/10.1145/3452412.3462753>), developed and maintained by WP2 partner BSC. Task T2.4 has two key objectives:

1. To support the development and enhancement of TALP by adding new features, extending its capabilities to GPU architectures and other stuff.
2. To integrate TALP with (selected) flagship codes and mini apps from WP2.

Task T2.4 is closely linked to the activities in T2.2 and T2.3. However, TALP is also expected to be utilized in other work packages, notably WP4, for certain Continuous Integration/Continuous Delivery (CI/CD) activities.

3. Methodology

In this section we will present the methodology used for the T2.2, T2.3 and T2.4

3.1 T2.2 Methodology

The methodology of task T2.2 can be summarized in four specific phases (Fig. 3-1). Each phase will be described in a more detailed way in the following sub-sections.

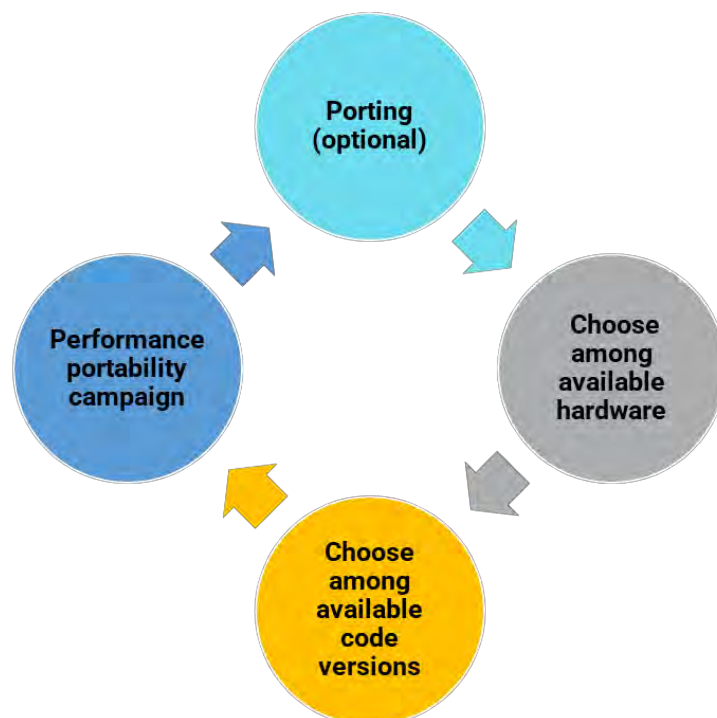


Fig. 3-1 the four steps in task T2.2's methodology.

3.1.1 Porting

This step of task T2.2 is highly important. It proposes creating new (with respect to a "baseline") portable versions according to the criteria outlined in section 2.1. The starting point can be the "full" application or an existing mini app, typically targeting a specific CPU and/or GPU flavour.

The porting methodology in T2.2 is further divided into two categories:

- Approach A (Maximum Performance), which aims to achieve maximum performances by using different accelerators. Naturally, this may lead to consider different software extensions for different devices. For example:
 - o NVIDIA: CUDA
 - o AMD: HIP
 - o Intel: SYCL

With this approach there might be issues with specialized external libraries (such as linear algebra).

- Approach B (Maximum Portability), which aims to achieve "acceptable" performance across different accelerators. The idea is to use a single software tool (e.g., OpenMP, SYCL) to support multiple devices. In this case, the challenge might result in suboptimal performances. Moreover, different implementations may still need to be considered for different devices.

Both approaches are valid and potentially usable in T2.2:

- Approach A offers high performance but suffers from low portability and is difficult to maintain or modify. Optimized code, while efficient, is inherently complex, making it unsuitable for frequent changes during scientific exploration.
- Approach B, on the other hand, provides reasonable performance, high portability, and ease of maintenance and modification, but sacrifices some computational efficiency.

Key Considerations:

The primary objective is to advance science, with flagship codes serving as tools for exploration. To achieve this, codes must be easy to maintain and modify, particularly as scientists often need to update the code themselves during their investigations. Approach A is less suited to this because highly optimized code tends to be inflexible.

Production Code Options:

For finalized production codes, there are two viable paths:

1. Allow scientists to make the necessary modifications themselves, prioritizing simplicity and ease of use over maximum performance.
2. Delegate the task to computer scientists or engineers, who can implement optimized versions of the new algorithms, ensuring efficiency while maintaining scientific input.

Ultimately, the choice between these approaches depends on the balance between performance requirements and the need for flexibility in scientific workflows.

3.1.2 Targeting EuroHPC and SDV accelerated systems

This task focuses on providing the necessary hardware for completing the PPC. Typically, this phase involves identifying the appropriate accelerated architecture, depending on the version or implementation of the available

mini apps. EuroHPC offers several dedicated calls to support this requirement (https://eurohpc-ju.europa.eu/access-our-supercomputers/eurohpc-access-calls_en).

For T2.2, it is recommended to use the hardware available through EuroHPC, but for the activities in T2.2 SDVs from vendors or project partners can also be employed to test and verify performance portability on architectures that are either not yet in production or are of particular interest to the project. CINECA has taken responsibility for managing this task on the project side. The full list of HPC architectures used in T2.2 is summarized in Appendix.

3.1.3 Targeting mini apps based on increased portability status

The porting activity aims to identify the available ecosystem for the PPC phase on the current hardware. Since porting is an ongoing process throughout the project, the decision on what to prioritize is based on the following heuristic criteria:

- Maximizing the variety of mini apps across different architectures, even if it may compromise performance.
- On the same platform, prioritizing portability by using different versions of a mini app (including different compiler versions, libraries, tools...).

3.1.4 Performance Portability Campaign

The PPC is the “core business” of task T2.2 and occurs after the phases outlined in sections 3.1.1, 3.1.2 and 3.1.3. The methodology involves running a given mini app (in the various GPU flavours described in section 2.1) on the selected architectures, following these progressively complex phases:

1. Portability testing (a maximum of 1 node is often used).
2. (Optional) Calculation of weak and strong scaling (where applicable).
3. (Optional) Performance optimization/tuning of the mini app on the given platform.

Both efficiency and performance portability metrics are computed based on the collected measurements in the above 1-3 phases. The calculation of performance portability metrics (both application-specific and eventually architectural) is performed on a limited set of resources and based on appropriate FOM following the working methodology described in the

ChEESE-2P tutorialⁱ (<https://digital.csic.es/handle/10261/369783>) and references contained within). Lastly, lessons learnt for the given flagship code are collected from the PPC outcomes.

3.1.5 T2.3 Methodology

The methodology of T2.3 is presented in the Fig. 3-2 below:

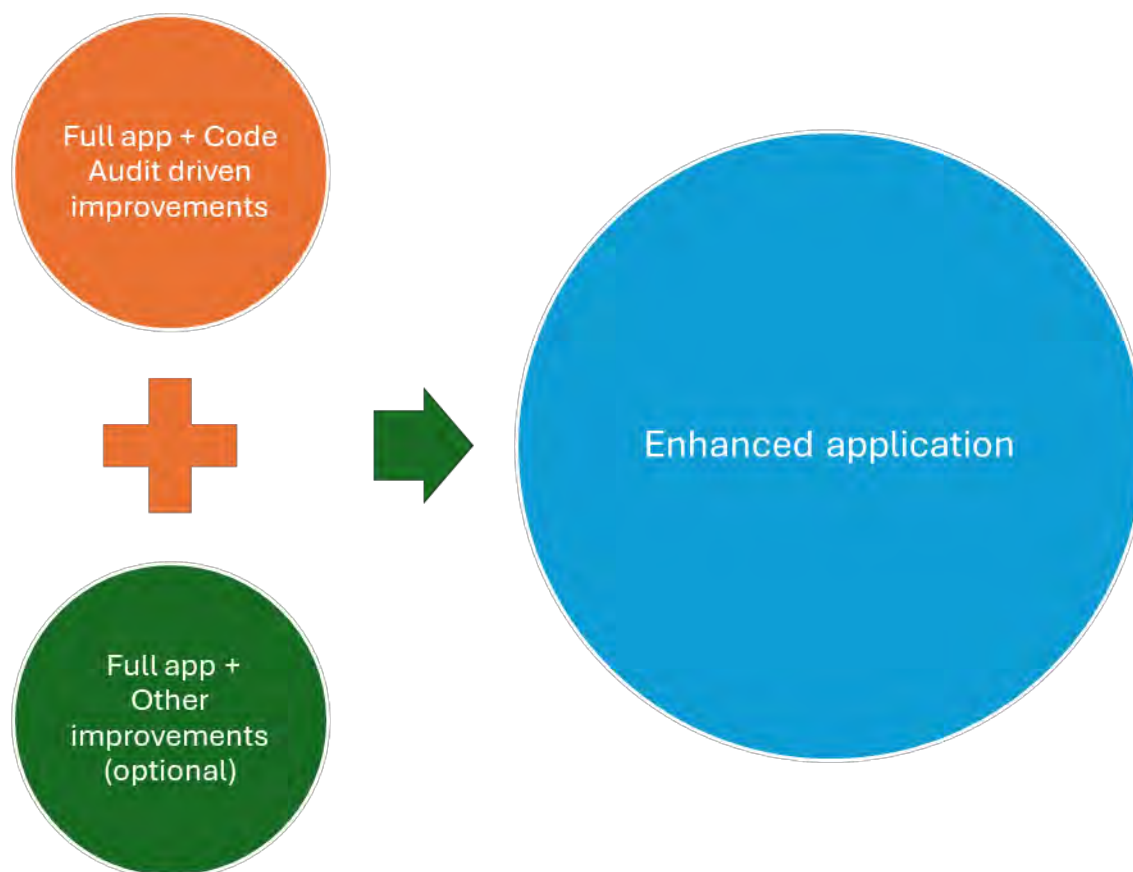


Fig. 3-2: the methodology of T2.3 (extended format)

The first macro-step involves an activity on the code utilized during the first code audit campaign (which include the code version and a specific programming language among those eventually to choose), applying the recommendations received and summarized in the deliverable D2.1.

The second macro-step (which can be done in parallel but is optional) is not assumed to operate on the same code version and is supposed to utilize recommendations and exploitation methods not directly related to the code audit. Of course, the final target of this version may differ (for example a direct transition from a CPU version to a GPU version is admissible).

It's important not to confuse this activity with the corresponding activity of task T2.2, as in that case the work is done on a mini-app, available or just ported to a GPU flavour.

3.2.1 Targeting EuroHPC general purpose systems

While task T2.2 focuses on accelerated architectures (whether from EuroHPC or elsewhere), task T2.3 expands the scope of investigation by considering both CPU and GPU-oriented architectures.

The decision not to consider SDV or similar technologies stems from the obvious consideration that, since T2.3 is the part of WP2 most oriented to Exascale, it is necessary for the target machine to meet specific criteria for stability and quality of service. This is generally achievable on the range of platforms provided in production by EuroHPC. The process that aims to identify the appropriate HPC architecture, depending on the version or implementation of the available full apps is the same of task T2.2 (see section 3.1.2).

3.2.2 Code Audit driven improvements

The code audit is one of the main tasks of the WP2, and the methodology is essentially the same as that inherited from ChEESE. The entire audit involves the participation of all ChEESE-2P codes in some flavour (CPU or GPU) of particular interest for the activities and a use case.

The audit campaign has produced outcomes both in terms of POP metrics and explicit suggestions on where to direct improvements in the specific version of the flagship code, in collaboration with POP (BSC and HLRS are project partners). The experience gained in the first audit campaign has been summarized in the deliverable D2.1.

The starting point for the T2.3 investigation (in terms of programming model, version, etc) for each code is reported in Tab. 3-1.

Code	Programing model	Version	System	Audit tool
SeisSol	MPI + SYCL + CUDA	1.1.3	Leonardo	Extræ; SeisSol self-profiling
SPECFEM3D	MPI + CUDA	devel 4.0.0	Leonardo	Scalasca/2.6.1; Score-P/8.3
ExaHyPE	MPI + OpenMP target	N.A.	Leonardo	Score-P/8.3
Tandem	MPI	1.1.0	LUMI-C	Score-P/8.3
xSHELLS	MPI + CUDA	N.A.	Vega-G	-
HySEA	MPI + CUDA	4.3.0	Leonardo	Extræ
FALL3D	MPI + OpenACC	8.2.1	Leonardo	Extræ and mpiP
OpenPDAC	MPI	1.0.0	LUMI-C	DLB/TALP and mpiP
LaMEM	MPI	N.A.	LUMI-C	Extræ
pTatin3D	MPI + HIP	N.A.	LUMI-G	Extræ
ELMER / ICE	MPI	fdd6c58f1	LUMI-C	Extræ

Tab. 3-1: flagship code, programming model, version, HPC machine and POP tool used for the code audit.

The methodology for the code audit driven improvements in T2.3 simply starts from codes as in the Tab.4-1 and relative practical recommendations from D2.1 and aims to produce a reasonable working plan, code by code along the reporting period.

3.2.3 Other improvements

There is a portion of the work in T2.3 that is not directly dependent on the code audit. This can occur for several reasons, including:

- The improvement may have already been planned elsewhere and is largely independent of the code audit outcomes.
- The version being investigated is not the one that was audited. For example, some code could decide to move to GPU from a CPU baseline or to a different GPU flavour. It's worthwhile to note that WP2 supports this type of investigation as it focuses on enhancing the portability (T2.2) and overall performance (T2.3) of each flagship code.

The methodology for Other improvements consolidates available actions from the above points and applies a similar approach used for code audit-driven improvements, addressing each code individually.

3.3 Methodology T2.4

The methodology of T2.4 is summarized as in the Fig. 3-3:

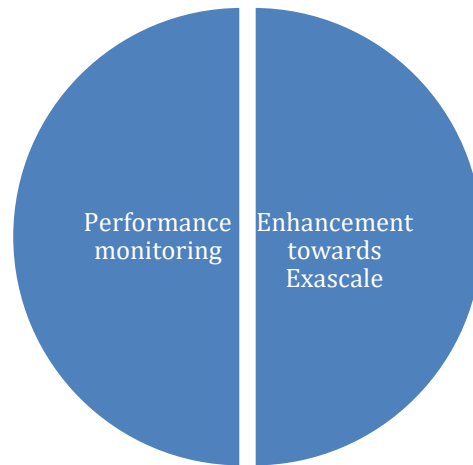


Fig. 3-3: the two step methodology of T2.4

More precisely, the methodology of task T2.4 from one side guarantee the correct evolution of tools for performance monitoring (real-time capable) and from the other side their development towards Exascale (complex and heterogeneous hardware and complex, highly parallel and GPU-accelerated software solutions).

3.3.1 TALP enhancement towards Exascale

The development of software for High-Performance Computing (HPC) systems presents several significant challenges. In recent years, the availability of computational resources has grown considerably, making it increasingly difficult for applications to scale effectively to Exascale levels. The key challenges lie not only in maintaining code correctness but also in avoiding performance degradation and scalability issues. To mitigate these risks, it is essential to have accessible feedback mechanisms that provide insights into how well an application is scaling.

At Exascale, tracing becomes difficult to manage due to the substantial memory and storage requirements. While benchmarking or measuring time to solution can provide some insights, they are insufficient for explaining the underlying reasons for an application's performance. What is needed is a solution that delivers insights into the behaviour of the application, with minimal impact on runtime and post-processing requirements.

TALP is a module within the Dynamic Load Balancing (DLB) library that has been developed with the objective of collecting performance metrics for MPI+OpenMP applications. As a low-overhead profiling tool, TALP enables the retrieval of performance metrics at the end of execution, as well as the ability to query them during runtime. The latest stable version of DLB includes support for POP-like metrics for the MPI model, with initial, experimental support for OpenMP metrics. Additionally, the integration of GPU performance metrics is also planned as part of the Task T2.4.

3.3.2 TALP performance monitoring

TALP provides developers with valuable feedback on the performance of their code. The reported metrics are a subset of the well-established POP performance metrics, which offer a quantitative analysis of key factors relevant to parallelisation. These metrics are calculated as efficiencies ranging from 0 to 1 in a hierarchical structure, with the top-level metric being "Global Efficiency," which is broken down into "Computation Efficiency" and "Parallel Efficiency," and further decomposed into finer-grained efficiencies.

The POP metrics reported by TALP for MPI and OpenMP applications offer valuable insights into potential scaling issues. One of the key advantages of TALP is that it requires no modifications to the application code. By intercepting MPI calls or acting as an OMPT tool for OpenMP, TALP automatically gathers event data from these programming models. However, if desired, developers can also use TALP's support for code annotations to specify regions for independent analysis. The following items are being implemented as part of T2.4:

MPI+OpenMP hybrid performance metrics

With DLB now supporting the OMPT interface to function as an OpenMP runtime monitoring tool, TALP can collect OpenMP performance metrics. Fig. 3-4 illustrates the hybrid model for MPI and OpenMP performance metrics (see deliverable D2.1 in the Appendix for an exhaustive explanation of the currently available set of POP metrics).

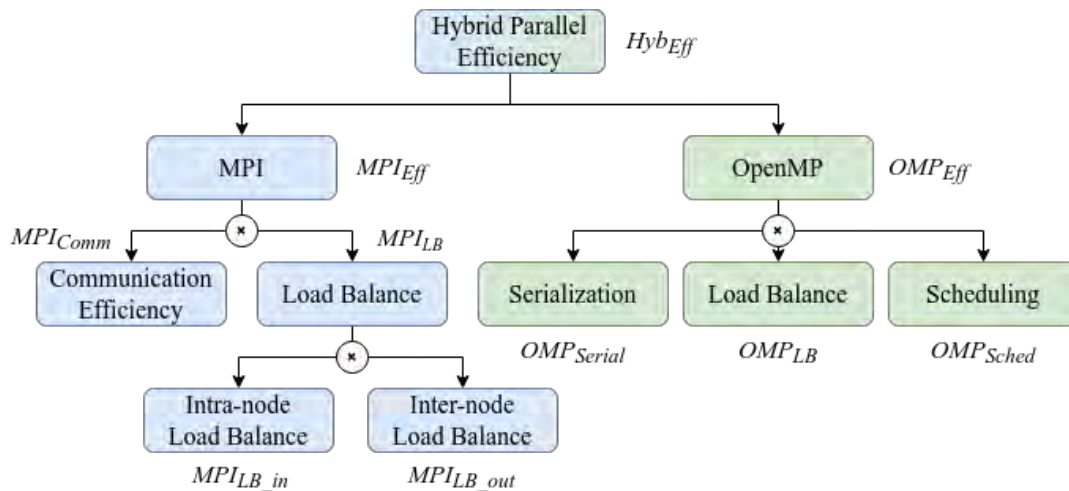


Fig. 3-4: POP hybrid model for MPI+OpenMP applications

In addition to the MPI performance metrics, TALP now also reports OpenMP performance metrics. The Hybrid Parallel Efficiency combines both the MPI Parallel Efficiency and the OpenMP Parallel Efficiency. The OpenMP Parallel Efficiency is derived from the following efficiencies:

- OpenMP Serialization: Represents the time when worker threads are inactive, either not executing implicit tasks (parallel region) or explicit tasks.
- OpenMP Load Balance: Measures the uneven distribution of execution time within parallel regions.
- OpenMP Scheduling: Accounts for performance penalties within parallel regions that are not related to load imbalance, such as runtime overhead, or reduced parallelism due to scheduling decisions or task dependencies.

TALP-Pages

TALP-Pages represents the next evolution in integrating Continuous Performance Monitoring into applications. This easy-to-integrate framework allows developers to quickly receive in-repository feedback on their code's performance.

TALP-Pages adds an extra layer by generating static websites to visualize the collected metrics and track their progression across different versions of the software. These self-contained HTML pages can be viewed independently or easily integrated into CI/CD solutions like GitLab Pages. TALP-Pages offers developers a simple way to monitor performance regressions and gain insights into code performance from execution data.

It is designed to operate within the development workflow, directly within the same repository, and without the need for additional hardware or resources. This is achieved by calculating performance metrics in real time using TALP and processing the generated JSON files. With TALP-Pages, we enhance existing frameworks by offering a solution that:

- Requires no additional hardware, aside from a runner with access to the target machine
- Provides valuable metrics for identifying scaling issues
- Combines regression analysis of performance metrics and time-to-solution in a single plot
- Delivers a tracing-free method to obtain a comprehensive table of POP-like metrics.

Fig.3-5 and 3-6 display the performance plots and scaling efficiency table generated by TALP-Pages.

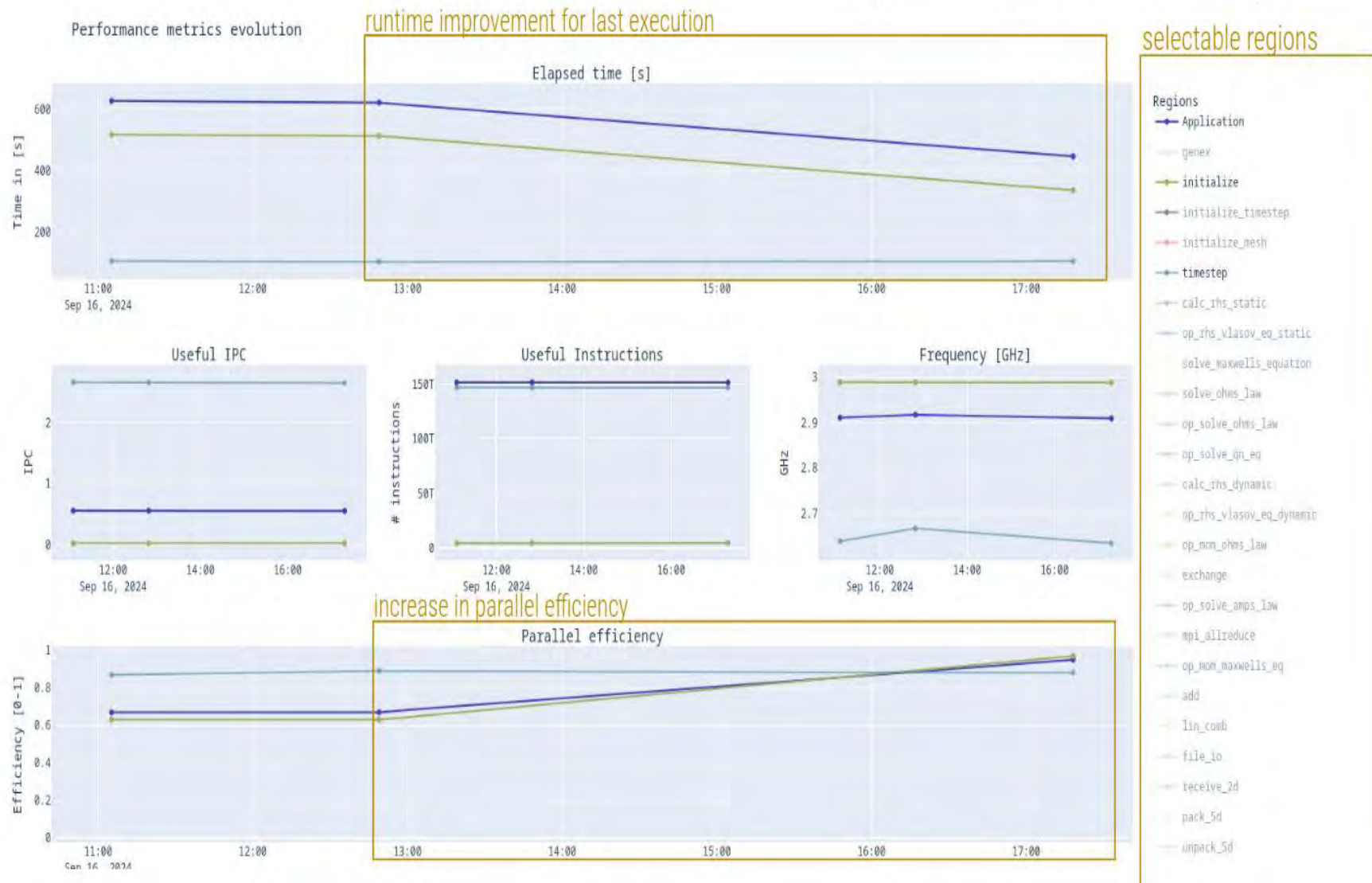


Fig. 3-5: Performance plots generated by TALP-Pages.

Region: timestep

Metrics	2xMPI 36xOpenMP	4xMPI 36xOpenMP
Global Efficiency	0.71	0.69
- Parallel efficiency	0.71	0.7
-- MPI Parallel efficiency	0.99	0.99
--- MPI Communication efficiency	1	1
--- MPI Load balance	0.99	0.99
---- MPI In-node load balance	0.99	0.99
---- MPI Inter-node load balance	1	1
-- OpenMP Parallel efficiency	0.71	0.7
--- OpenMP Scheduling efficiency	0.99	0.99
--- OpenMP Load balance	0.99	0.99
--- OpenMP Serialization efficiency	0.73	0.72
- Computation Scalability	1	0.98
-- Instructions scaling	1	0.99
-- IPC scaling	1	0.99
-- Frequency scaling	1	1
Useful IPC	2.22	2.19
Frequency [GHz]	2.36	2.36
Elapsed time [s]	20.35	10.55

Fig. 3-6: Scaling efficiency generated by TALP-Pages.

4 Flagship codes

This section presents the results achieved in the deliverable, organized code by code. Each section provides the following subsections:

1. Flagship Code/Mini App(s) Initial Status: This outlines the starting point of the flagship code, including any associated mini-apps and their initial status.
2. Porting Activities: Describes the (optional) porting activities that may have facilitated the development of new mini apps as part of Task T2.2.
3. Performance Portability Campaign: Details all the steps undertaken to measure the current performance portability (refer to section 3.1.4 and related documentation within).
4. Code Audit-Driven Improvements and 5. Other Improvements: Summarizes enhancements implemented as part of Task T2.3, including improvements identified through code audits and other improvements.
6. Exascale Ambition: Explains how the improvements described in the points above from 1 to 5 align with and support the strategic objectives and project goals towards exascale computing. It's important to remark that both Capability (problems that traditionally have been parameterised because are unaffordable with current hardware), Capacity (ensembles of single problems affordable with petascale-range machines but that can aggregate into an exascale workflow) and Urgent Computing use cases are considered.

The contents for each section and code vary in length and detail. This, ultimately, depends on the state of the code (and related mini apps), the effort invested by the code owners and collaborators in WP2, and the other partners' support for a given task related to that specific code.

4.1 SeisSol

SeisSol simulates extreme-scale earthquake dynamic rupture scenarios using high-order discontinuous Galerkin (DG) discretization on unstructured adaptive tetrahedral meshes and Arbitrary high-order DERivative (ADER) time stepping. In the CPU version, it supports seismic wave propagation for various rheology (elastic, anisotropic-elastic, with

plastic deformation; viscoelastic, poroelastic, coupled elastic-acoustic) and simulates the dynamic rupture process on prescribed faults (incl. faults with complicated geometries, splay faults, etc.) via a wide range of supported friction laws. Several petascale production runs have been performed on generations of supercomputers, including Tianhe-2, Cori, Shaheen-II, SuperMUC-NG and Frontier.

4.1.1 Flagship code/Mini-app(s) initial status

As part of the ChEESE project, the basic functionality of SeisSol was ported to GPU architectures, focusing on NVIDIA GPUs and CUDA as the primary programming model. Porting was achieved for elastic/elastic-acoustic wave propagation kernels and for the friction law required for the pilot demonstrator scenario in ChEESE-1P, enabling 3D fully coupled simulation of earthquake-tsunami events. Important insight obtained during ChEESE included memory-bound performance of SeisSol on GPUs (vs. clearly compute-bound performance on CPUs), due to restrictions in the original CPU-inspired code generation approach (focusing on chains of small element-local matrix multiplication) and identifying scalability challenges resulting from SeisSol's cluster-oriented Local Time Stepping (LTS) algorithm (Fig. 4-1.1).

For performance optimization of fundamental compute kernels, SeisSol provides the *SeisSol-Proxy*, a mini app that executes SeisSol's element-local update kernels on random data. SeisSol-Proxy is used for first steps in porting as well as for performance optimization of kernels. Increasing the element count for SeisSol-Proxy reveals the performance dependency of SeisSol for different sizes of LTS clusters:

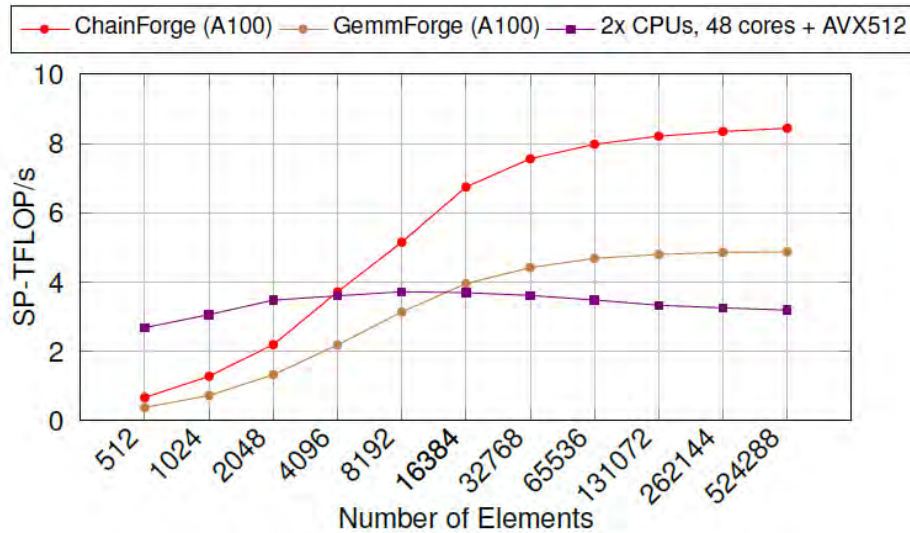


Fig. 4-1.1 Single-precision performance (on a single NVIDIA A100 GPU) of SeisSol-Proxy for increasing number of elements, comparing two developing stages of the matrix multiplication backends (ChainForge vs. GemmForge) and performance obtained on a CPU node.

4.1.2 Porting

For ChEESE-2P, there are two general goals regarding the GPU performance of SeisSol:

1. Extend the functionality of SeisSol on GPUs to more rheologies and rupture laws, aiming towards a multi-physics earthquake simulation quake on GPU architectures.
2. Tackle the performance limitations identified in ChEESE (section 4.1.5).

We have extended the GPU port for anisotropic-elastic material and for elastic material with plastic deformation. Both are part of the released version of SeisSol. Viscoelastic material on GPUs requires support for small element-local tensor operations (in contrast to matrix operations for the simpler models), which requires support by SeisSol code generation backends (see below). Viscoelastic wave propagation is already functional for GPUs and will be released in the second half of ChEESE-2P, after further testing and performance optimization. In terms of dynamic rupture simulation, SeisSol now supports all relevant friction laws on GPU, due to a redesign of the respective dynamic rupture implementation, based on SYCL. SYCL (<https://www.khronos.org/sycl/>) is an open standard developed by the Khronos Group that allows a single code to compile and run for several architectures (CPU, GPU, FPGA). We have also ported point sources to allow for kinematic earthquake rupture sources (required in probabilistic hazard assessment and for inverse problems) and the gravity boundary condition to correctly capture the onset of tsunami propagation in the fully coupled elastic-acoustic earthquake-tsunami model. Moreover,

we have extended SeisSol's GPU port by providing HIP/ROCm kernels for AMD GPUs (SYCL is used for dynamic rupture kernels; OpenMP is experimented as part of a separate project). We have also continued the work started in ChEESE-1P to optimize for Arm CPUs towards using NVIDIA Grace architectures.

4.1.3 Performance Portability campaign

For the portability issues we have successfully installed and tested, or are testing, SeisSol on the following platforms:

- **Leonardo (NVIDIA A100 GPUs, "Ampere")**, as part of the first ChEESE-2P audit campaign
- LUMI-G (AMD Instinct MI250X GPUs), also as part of a parallel subproject within the EuroHPC JU project Inno4scale.
- MareNostrum **5 (NVIDIA H100 GPUs, "Hopper")**, as preparation for the second audit campaign (see section 4.1.4).
- Vista (NVIDIA GH200 "superchip", "Grace Hopper"), as part of a collaboration with the Texas Advanced Computing Centre.
- SuperMUC-NG phase 2 (Intel Data Center GPU Max 1550, "Ponte Vecchio"): Preliminary experiments have also been started.

Fig. 4-1.2 shows a comparison of SeisSol-Proxy performance between A100 and MI250X GPUs; Fig. 4-1.3 shows a corresponding plot for the GH200 "superchip". Fig. 4-1.3 particularly reveals the launch overhead of kernels, as well as the lack of parallelism for small LTS cluster sizes as the key performance limitations.

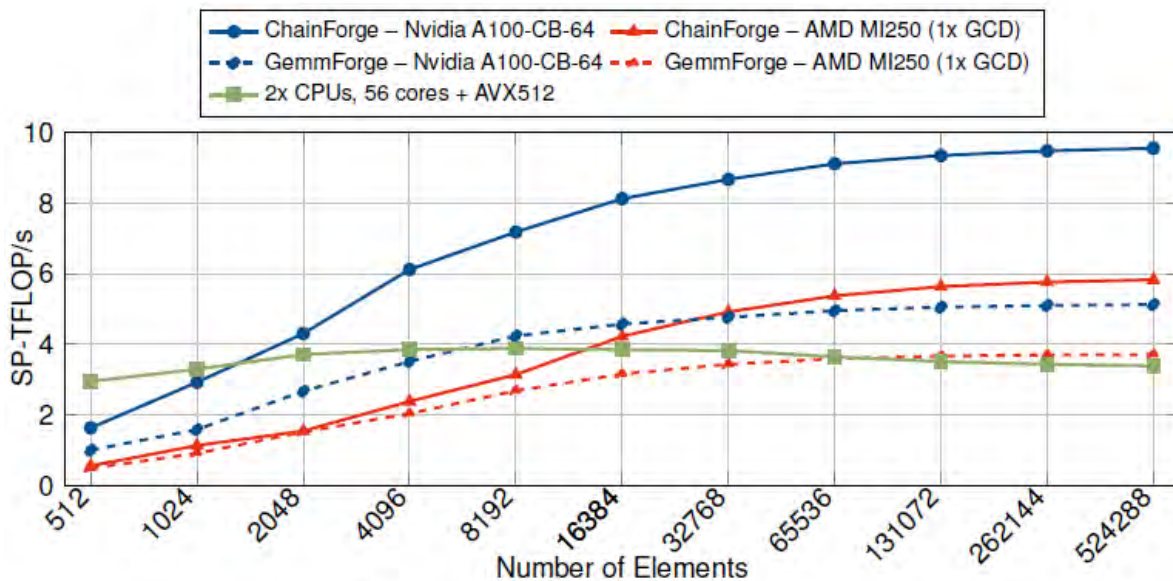


Fig. 4-1.2: Single-precision performance of SeisSol-Proxy for increasing number of elements, comparing A100 and MI250X (results published in Dorozhinskii et al., 2024 <https://doi.org/10.1002/cpe.8037>).

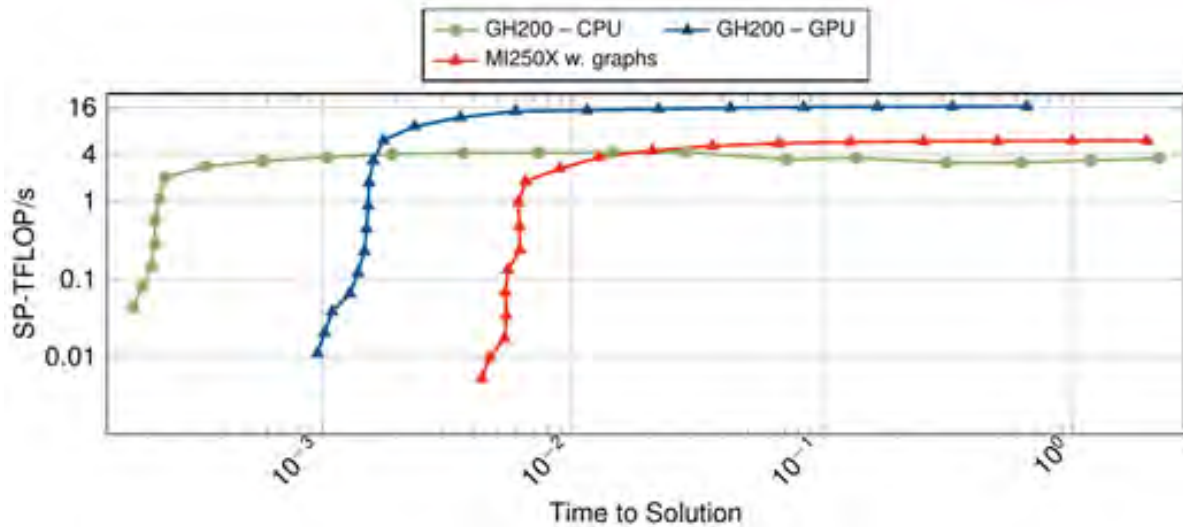


Fig. 4-1.3: Single-precision performance of SeisSol-Proxy for increasing number of elements, comparing the Grace Hopper GH200 “superchip” (CPU and GPU) with MI250x. Note that the plot gives execution time on the x-axis and thus reveals the launch overhead of CPU vs. GPU (esp. for MI250X), and its substantial impact on performance for small LTS clusters.

SeisSol relies on CUDA graphs to efficiently launch the GPU kernels for updating the LTS clusters. The respective functionality in ROCm has turned out to be much less mature, and only recently (with version 6.2) has reached the functionality assumed by the SeisSol implementation. The situation is made more complicated by the software stacks offered on supercomputers – LUMI, for example, is only at version 6.0.3. even after an update of the software stack in September 2024. Considering the achieved maximum performance, we can compute the Relative BW Efficiency for the 3 architectures (Leonardo, LUMI-G and Vista).

Architecture/GPU	Software stack	Max TFLOP/s	Theor. Available BW (TB/s)	Relative BW Eff.
NVIDIA A100	CUDA 12.1	9.56	1.64	5.82
NVIDIA GH200 (CPU)	CUDA 12.3	3.52	0.5	7.04
NVIDIA GH200 (GPU)	CUDA 12.3	16.46	4.0	4.11
AMD MI250X	ROCm 6.2	5.98	1.6	3.73

Tab. 4-1.1: Maximum performance (single-precision TFLOP/s) achieved with SeisSol-Proxy for the elastic wave propagation kernel. The "Relative Bandwidth Efficiency" is the ratio of max. TFLOP/s vs. maximum (theoretically) available bandwidth (given in the third column). Higher is better.

4.1.4 Code Audit driven improvements

As reported in Deliverable D2.1, the first code audit with SeisSol on Leonardo (using Extrae and Score-P) was limited by incapacibilities of the profiling tools to cope with SeisSol's mixed threading and MPI communication model (using a Posix communication thread to overlap communication with computation). In cooperation with BSC, we have successfully installed SeisSol on MareNostrum 5. BSC has obtained first Extrae traces for the small benchmark scenario, such that we are positive to be able to produce a full POP3 audit in the second audit campaign.

4.1.5 Other improvements

SeisSol uses its code generator, *YATeTo*, to turn the complex ADER-DG element-update kernels (expressed via the *YATeTo* DSL) into a sequence of small matrix operations, which on CPUs are executed by highly optimized small-BLAS libraries. In ChEESE, we mapped this approach to GPUs by launching batches of small BLAS operations to the GPU, using custom GPU back-ends: *GemmForge* was limited to binary matrix multiplications, as provided by *YATeTo*; *ChainForge* detects chains of matrix multiplications and launches batched kernels that compute the chain product. While *ChainForge* achieves a substantial performance gain over *GemmForge*, each element update still requires multiple GPU kernel launches, which limits the arithmetic intensity of each kernel and causes memory-bound performance (cf. Dorozhinskii et al., 2024 <https://doi.org/10.1002/cpe.8037>). We have therefore updated *YATeTo* to generate small tensor operations, and are implementing a respective GPU backend, *TensorForge*, to execute batches of such tensor operations.

4.1.6 Exascale ambition

SeisSol will be used within the Probabilistic Seismic Hazard Analysis (PSHA) and Bayesian inversion workflows developed within PD1, esp. simulation case SC1.3 (Physics-based PSHA and urgent UQ for Iceland). Here, we envisage a capacity computing use case, with SeisSol forward models running on 10 to few 100 of nodes. For SC1.2 (Next generation of earthquake simulators for the Hellenic arc and Iceland) we execute 3D fully coupled elastic-acoustic models, exploiting simulated subduction zone data provided by other flagship codes. Similar fully coupled models are envisaged for PD4 (Complex multi-source tsunami modelling). These fully coupled models lead to the largest models currently simulated with SeisSol

(more than 500 million of elements, towards 10^{12} degrees of freedom) and may require beyond 1000 nodes of the current pre-exascale machines (LUMI, Leonardo, etc.) for a single run.

As part of a collaboration with the Texas Advanced Computing Center, we have participated in several “Texascale Days” and have performed scalability tests and production runs for several dynamic rupture scenarios on the Frontier¹ supercomputer, up to full machine size. This includes:

- Three production runs on the 2023 Kahramanmaraş Earthquake (Zhe Jia et al., 2023 <https://doi.org/10.1126/science.adi0685>), achieving around 10 PFLOP/s on 8192 nodes.
- A performance comparison of several setups for the 2019 Ridgecrest Earthquake (Taufiqurrahman, 2023 <https://doi.org/10.1038/s41586-023-05985-x>), using 4 different meshes with between 178 and 620 million grid cells, achieving up to 9.2 PFLOP/s on 8170 nodes and revealing a strong performance impact of the fault geometry (number of elements at the fault).
- A scalability study on the local time stepping algorithm, for the 3D fully coupled elastic-acoustic scenario for the 2018 Palu, Sulawesi Earthquake and Tsunami (Krenz et al., 2023 <https://dx.doi.org/10.1145/3458817.3476173>) for two meshes with 89 and 512 million elements, on up to 8000 nodes.

4.2 SPECFEM3D

SPECFEM3D is an open-source high-performance computing (HPC) code designed for simulating the propagation of mechanical waves in fluid and solid media. It accommodates various physical laws, including elastic and viscoelastic models, for isotropic and anisotropic materials. These capabilities make it a critical tool in seismic wave propagation and related fields.

4.2.1 Flagship code/Mini-app(s) initial status

The codebase is primarily written in Fortran, with optimisation for CPUs. In the early 2010, SPECFEM3D was extended to support GPUs through CUDA, which has enabled significant performance improvements on NVIDIA GPU architectures. More recently, as part of the European ChEESE Center of Excellence initiatives, support was added for AMD GPUs using HIP, which

¹ It is the #2 supercomputer of Top500. It is based on AMD GPUs.

has enhanced its portability and ensured compatibility with heterogeneous HPC systems. Compilation targets specific architectures at build time, leveraging *nvcc* for NVIDIA GPUs, *hipcc* for AMD GPUs, and traditional Fortran compilers for CPU-only environments.

SPECFEM3D employs a domain decomposition approach for parallelization, utilising MPI for inter-process communication. Each sub-domain is associated with a specific MPI rank, assigned to either a CPU or a GPU. The computations for each sub-domain are performed independently, with neighbouring sub-domains exchanging boundary data via peer-to-peer non-blocking communication (*MPI_Isend* and *MPI_Irecv*). However, communication between GPUs is done via the CPU memory to call MPI functions.

4.2.2 Porting

SPECFEM3D had been ported to NVIDIA and AMD CPUs and GPUs. Porting activity required minimal additional work. The methodology recommended in WP2 was followed, whereby mini apps were developed for each CPU, GPU NVIDIA and GPU AMD architecture by extracting the SPECFEM3D calculation kernels for different physics, namely fluid, isotropic elastics, anisotropic elastics and viscoelastic.

4.2.3 Performance Portability campaign

The performance portability campaign was based on the utilisation of a variant of the mini app, namely the isotropic elastic kernel (https://gitlab.com/specfem_cheese_2p/mini_specfem/specfem_mini_app).

Tests were conducted on two NVIDIA systems: the Leonardo supercomputer equipped with custom A100 GPUs and a cluster named Thea equipped with a GH200 “superchip” (<https://resources.nvidia.com/en-us-grace-cpu/grace-hopper-superchip>) provided by NVIDIA.

Both the CUDA and HIP versions of the mini application were executed, each targeting a single GPU. Additionally, tests were performed on ADAstra, an HPE-Cray French supercomputer hosted at CINES (<https://dci.dci-gitlab.cines.fr/webextranet/architecture/index.html>) equipped with AMD MI250X accelerators. In this case, the HIP version of the mini application was used to target a single Graphics Compute Die (GCD).

	Leonardo (Custom A100)		Thea (GH200)		Adastra (MI250X)		PP metric
Version	Time Loop (s)	App. Eff.	Time Loop (s)	App. Eff.	Time Loop (s)	App. Eff.	
CUDA	8.56	100%	3.85	100%	N.A.	N.A.	0
HIP	9.82	87%	4.41	87%	4.5	100%	0.91

Tab. 4-2.1 FOM, Application Efficiency and PP metric for CUDA and HIP version

We conduct our analysis by considering the Figure of Merit (FOM) as the time spent in the main iteration loop. Tab. 4-2.1 presents the results in terms of Application Efficiency and associated PP metric for this FOM for both the CUDA and HIP versions across the target systems, averaged over five independent runs for statistical significance.

To assess the performance portability of the mini application, we first calculate the application efficiency for each configuration. The application efficiency indicates the percentage of the optimal performance achieved by the application on a given platform. We compute it by considering the best timing on each platform as an upper bound on the attainable peak performance. For instance, the HIP version operates at 87% efficiency on both the Leonardo and Thea platforms compared to the best runtime achieved by the CUDA version. This reduced efficiency relative to the more optimal CUDA version can be attributed to the additional overhead introduced by the HIP-to-CUDA function calls (<https://gpuopen.com/learn/amd-lab-notes/amd-lab-notes-hipify-readme/#general-tips>).

Using the calculated application efficiencies, we then compute the performance portability metric allowing us to quantify the trade-off between performance and portability across multiple platforms. According to this metric, the HIP version achieves the highest performance portability given that the CUDA version is restricted to running on NVIDIA GPUs.

It is important to point out that the baseline that we have considered as peak attainable performance may not coincide with the actual peak value achievable on the platform, as the CUDA version of the mini app might still benefit from further optimisations. Future PPC could consider Architectural efficiency (as defined in this internal WP2 tutorial <https://digital.csic.es/handle/10261/369783>) and other Platforms and GPU flavors.

4.2.4 Code Audit driven improvements

Following the initial code audit, it was observed that there was a decrease in efficiency in strong scaling. The communication between GPUs reduced the speed of the code when 512 GPUs were utilised in the test case. Peer-to-Peer MPI communications between GPUs could no longer be fully overlapped by computation despite the use of non-blocking communications. To address this, MPI CUDA-aware was employed to facilitate direct communication between GPU memory pointers. This

approach led to an improvement in scalability during tests conducted on Leonardo (Fig. 4-2.1)

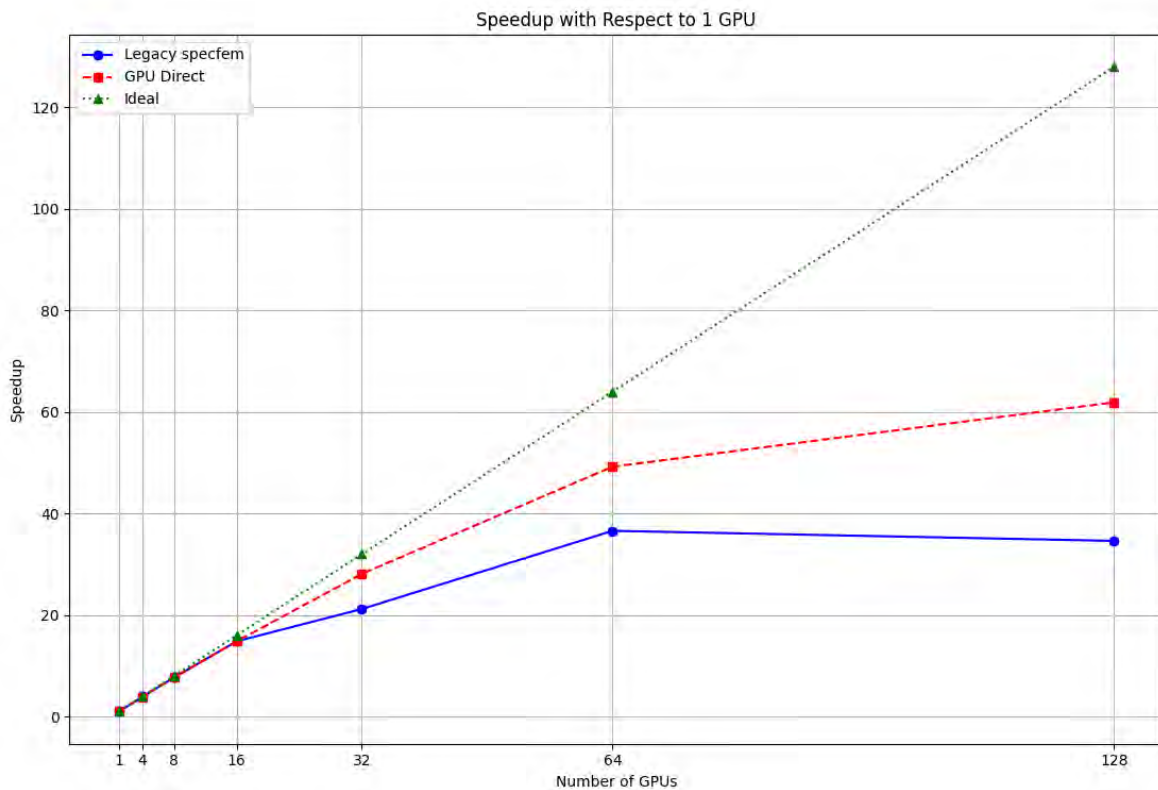


Fig. 4-2.1 Strong scaling in Leonardo-Booster. Comparison with legacy SPECFEM3D version (MPI communication using CPU memory) and MPI-CUDA aware communication.

4.2.5 Other improvements

SPECFEM3D is a computationally intensive, memory-bound code, making memory bandwidth a performance bottleneck. To address this, we investigated the use of lossy compression and decompression techniques to increase the effective memory bandwidth. While these methods introduce a controlled loss of precision, this trade-off is acceptable for our application, as the data used to generate the model properties is known to within a few percent. We have found that maintaining a relative residual between original and encoded data within 1% of tolerance is acceptable for our use case. We explored multiple approaches, including wavelet compression, jpeg inspired compression, AVX2-optimized fixed-point codecs (ranging from 4 to 16 bits), trilinear interpolation, and 16-bit floating-point storage. A test platform was developed to benchmark these codecs on numerical accuracy and performance across platforms like AMD Genoa/Milan and Intel Sapphire Rapids. The results showed significant improvements in effective bandwidth, with decompression speedups ranging from 3x to 7x and compression speedups of 2x. Trilinear

interpolation proved to be highly efficient, especially for smooth models. Fixed-point codecs and floating-point variants showed a good balance between speed and numerical accuracy. Future steps include integrating these codecs into SPECfEM3D mini apps to evaluate their impact on simulation performance and accuracy. This approach aims to overcome bandwidth limitations, enabling faster and more scalable computations in SPECfEM3D.

4.2.6 Exascale ambition

In the CHEESE-2P project, the SPECfEM3D code is involved in Simulation Case 1.1 (which is a potential Scientific Grand Challenge), which plans to perform a high-frequency (100 Hz) seism-acoustic simulation. The environment consists of highly attenuating unconsolidated sediment, a layer of water and deeper geological structures with higher seismic velocities. These characteristics imply that the mesh size required to reach 100 Hz must include a billion elements. So far, we have been able to run a simulation with 100 million of elements on 1024 V100 GPUs on the Marconi100 machine. Our ambition is to be able to handle a mesh that is an order of magnitude larger. This is a first step because, from a scientific point of view, we need to be able to run simulations at several hundred Hz, so Exascale machines will be essential to this purpose.

4.3 ExaHyPE

ExaHyPE 2 is an engine to solve problems formulated via hyperbolic PDE systems, using high-order discontinuous Galerkin (DG) discretisation with Arbitrary high-order DERivative (ADER) time stepping, on dynamically adaptive tree-structured Cartesian meshes. In addition, Finite-Volume solvers can be used, both as stand-alone solvers and as a-posteriori limiter for the ADER-DG solver. Scalability limitations identified during ChEESE project have motivated a rewrite of ExaHyPE and its underlying Peano adaptive AMR framework towards ExaHyPE 2, which is the basis in ChEESE-2P. ExaHyPE/ExaHyPE 2 are co-developed by TUM and the group of Tobias Weinzierl (Durham University). Porting and optimizing ExaHyPE 2 for GPU architectures is a central goal of WP2.

4.3.1 Flagship code/Mini-app(s) initial status

The initial status of ExaHyPE 2 was a CPU-only implementation with no support for GPU accelerators. The treatment of dynamic AMR in ExaHyPE 2 has motivated a complicated task-based approach for spacetree traversals, employing a visitor pattern.

A first attempt of porting ExaSeis, our package to solve seismic wave propagation with dynamic rupture sources, to GPU architectures has been done using so-called enclave solvers with task fusion (Wille et al., 2023 https://link.springer.com/chapter/10.1007/978-3-031-32041-5_4).

Enclave tasks consist of updates to elements that are not subject to strict dependencies due to inter-node communication or refinement boundaries. Such tasks are fused on-the-fly and offloaded to GPU (using OpenMP target offload). The first ChEESE-2P code audit on ExaSeis identified bottlenecks concerning task orchestration and overhead in our load balancing that effectively prevented scaling of the code. Challenges to optimize the parallelization of ExaHyPE 2 lie in the spacetree-splitting approach (domain decomposition) of Peano, the underlying AMR framework, and improving the scheduling algorithms of enclave tasks and their fusion within ExaHyPE 2. No mini app was available in the initial status of ExaHyPE 2.

4.3.2 Porting

Our GPU port uses OpenMP target offloading for the numerical kernels. We have first implemented kernel benchmarks that employ static enclave task fusion on a Cartesian grid without AMR or load balancing. As ExaHyPE 2 is an engine with a sophisticated code generator, designed for implementing a variety of applications and PDEs, these kernel benchmarks can be used to analyse the performance of any type of PDE on the CPU as well as on the GPU. Correctness checks have also been implemented (such as comparing the outcomes of a CPU kernel and a GPU kernel). We consider this as our mini app, as the kernel benchmarks strip down most of ExaHyPE 2's complicated features (such as dynamic AMR); nevertheless, with the mini app we evaluate the pure numeric that is also used in the full applications of ExaHyPE 2. This means we can more easily identify performance bottlenecks in production code.

Our first attempt to port our kernels using OpenMP target offload revealed significant performance challenges across all major GPU architectures, due to too fine-grain kernels.

We therefore implemented some of our kernels using CUDA. While the CUDA implementation will likely not enter the main branch, due to lack of performance portability, they provided valuable insight. It delivered a substantial performance improvement, highlighting optimizations e.g. specific to memory management that were not effectively leveraged in the

OpenMP implementation due to OpenMP's nature to hide everything in pragmas.

With these insights, we revised our OpenMP kernels, changing to explicit memory handling, and thus removing several of the pragmas (map clauses) we relied on initially. Furthermore, with more profiling and support by NVIDIA, we could improve the performance by optimizing memory layouts and avoiding unnecessary roundtrips of memory in our kernels (decreasing the number of loops).

While we see that our pure numerical kernel implementations perform well on different GPU architectures using our mini app (section 4.3.3), we still encounter the pre-existing bottlenecks in task orchestration and load balancing when using a full app in ExaHyPE 2 (section 4.3.4). These bottlenecks significantly reduce the speedup when using an accelerator in a full simulation due to overhead on the CPU; the GPU does not receive data fast enough to fully utilize the hardware (will be main topic for the second code audit).

4.3.3 Performance Portability campaign

The revised OpenMP target version of our mini app has been tested on NVIDIA A100 and H100 GPUs, and on Intel Data Center GPU Max 1550 GPUs. First results of our improved OpenMP target finite volumes kernels are shown in fig. 4-3.1:

Hardware Platform	Code Support/Toolchain
Leonardo (A100, 2000 GB/s)	NVHPC, LLVM Clang, oneAPI + Codeplay
Claix (H100, 2400 GB/s)	NVHPC, LLVM Clang
SuperMUC-NG Phase 2 (Max1550, 3276.8 GB/s)	Intel oneAPI

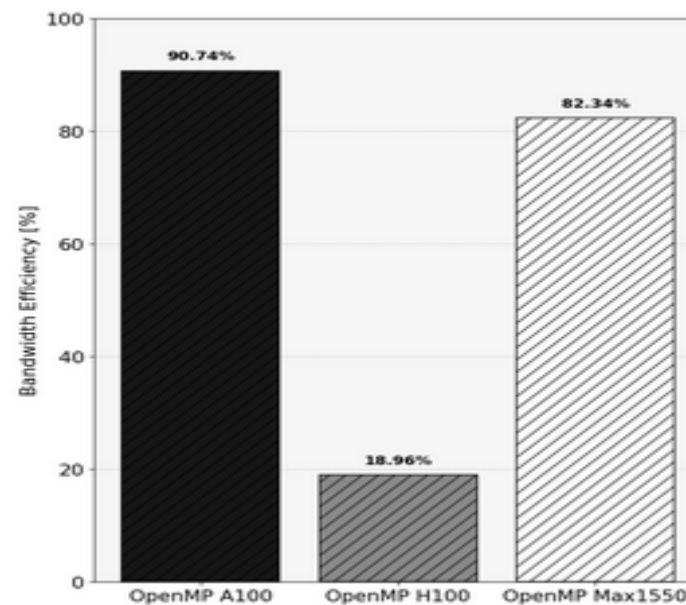


Fig. 4-3.1: Tested Platforms and Toolchains (left) and percentage of Bandwidth Efficiency (right). Memory-bound regime, single GPU.

It's worthwhile to note that the performance on the H100 does not reflect the true performance of the ExaHyPE 2 mini app. We needed to fall back to a less optimized compiler version as the LLVM module was non-functional at the time of testing.

Further benchmarking and collecting more results for our ADER-DG kernels will be one of the next activities for our mini app as well as the comparison of different programming models (OpenMP vs. SYCL vs. *std::par*) and using latest GPU architectures (AMD MI300A, NVIDIA GH200). SYCL and *std::par* counterparts to our OpenMP target kernels already exist for the finite volumes kernels and are under development for ADER-DG.

4.3.4 Code Audit driven improvements

For the first ChEESE-2P code audit we considered the LOH1 benchmark as implemented in the ExaSeis package. LOH1 is a common benchmark to verify correctness of seismic wave propagation. The uniform Cartesian grid chosen for this benchmark contained 531441 cells with a discretization order of 7. At the time of the first audit, the comparably low number of cells was restricted due to a bug in our load balancing. This has been resolved in new iterations of the code; therefore, we are now able to create significantly larger meshes. Still, the load balancing remains suboptimal when using static or dynamic AMR.

One major bottleneck, as identified by the first code audit, is Peano's spacetree parallelization: Each core triggers a spacetree split in Peano, thus creates a new subdomain including overhead introduced by ghost cells. ADER-DG's high-order discretisation tends towards few elements per core, such that on CPUs with many cores, the number of ghost cells increases to an extent that it causes substantial additional memory usage and memory transfers, thus preventing strong scaling of the code. Furthermore, we see that the load balancer can perfectly balance the load of the inner domain cells across the spacetrees, however, it does not consider the ghost cells. Effectively, this means that there are spacetree-traversal tasks that need longer than other traversal tasks to finish one grid-sweep.

This results in unnecessary idling of cores as we can see in the Vampir trace (Fig. 4-3.2).

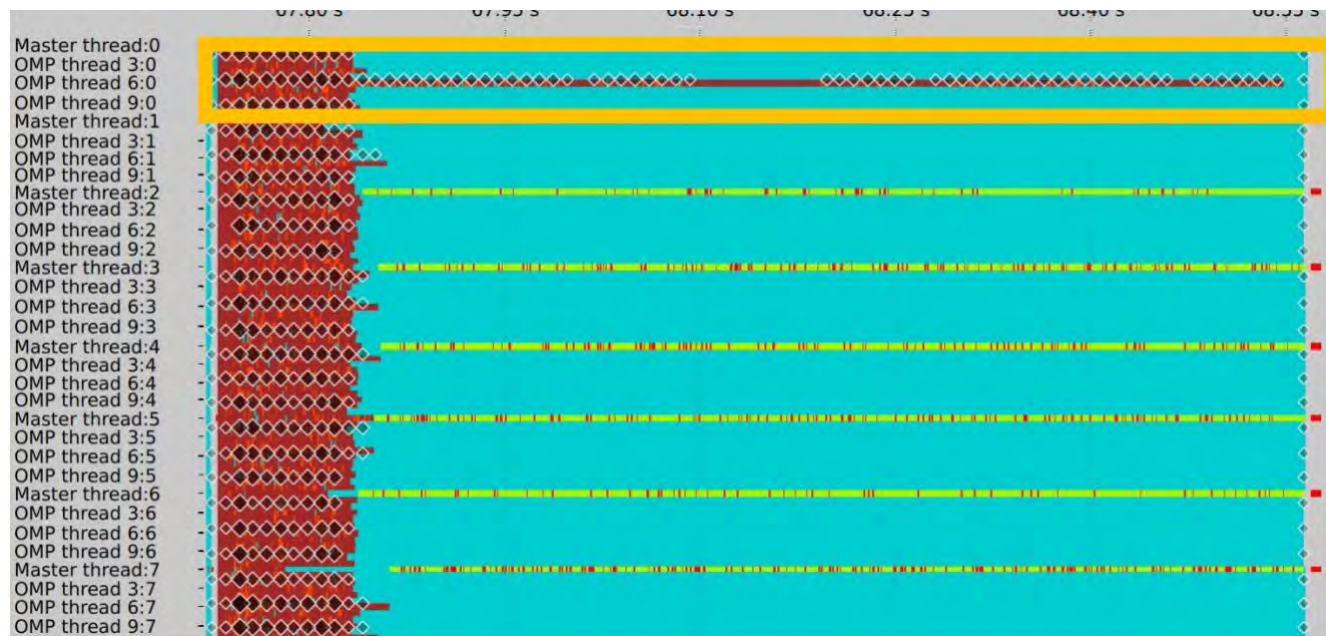


Fig. 4-3.2: Vampir trace output

This trace shows a setup with 8 MPI ranks and 4 OpenMP threads (spacetree-traversal tasks) per rank. We see that one of the trees in rank 0 needs excessively longer time to traverse its subdomain (depicted as red) while all other cores are idling (depicted as blue). However, the spacetrees are perfectly load balanced (of equal size), but the tree in rank 0 carries a greater amount of ghost cells, and thus, has higher overhead.

Enclave tasking remedies this by spawning additional independent worker tasks, aiming to be picked by the idling cores (fine-grained load balancing). For this, fine-tuning is required to find the optimal number of spacetrees per rank, utilizing some CPU cores for only grid traversals, while all remaining CPU cores pick enclave tasks for computation. For GPU offloading, we fuse individual enclave tasks to a “meta task” that is offloaded to the GPU. For high performance, we need to identify enclave cells as fast as possible on the CPU – by increasing the number of spacetrees per rank, we try to utilize all available CPU cores to identify the enclave cells for fusion.

First improvements have been implemented to avoid creating ghost cells for synchronization within one rank. They are instead created when we need to distribute data across rank boundaries, sharing data between spacetrees on one rank uses shared memory instead. Task orchestration and fusion strategies for GPU offloading have been improved; still with potential for more optimizations. Further optimizations in load balancing are ongoing as well. Some minor improvements have been implemented as suggested by the first code audit: we restricted the number of MPI communicators created per rank to the actual number of spacetrees that are created. In the previous version, we had created a fixed number of 1024 MPI communicators per rank, resulting in very high overhead of MPI communication for many ranks.

4.3.5 Other improvements

No Other improvements for ExaHyPE in the reporting period.

4.3.6 Exascale ambition

The ExaSeis seismic wave propagation solver shall be used within the probabilistic seismic hazard (PSHA) and Bayesian inversion workflows developed within Pilot Demonstrators PD1, esp. simulation cases SC1.3 (Physics-based PSHA and urgent UQ for Iceland). Here, we envisage a capacity computing use case, with ExaSeis forward models running on 10 to 100 nodes. For PD4 (Complex multi-source tsunami modelling), a landslide model is developed within WP5. Due to the 2D nature of this

model, we expect it to run on single or few compute nodes, as part of workflows for simulating coupled events.

4.4 Tandem

Tandem is a scalable discontinuous Galerkin code designed for solving linear elasticity problems and modeling sequences of earthquakes and aseismic slip on unstructured curvilinear grids (Uphoff et al., 2023 <https://doi.org/10.1093/gji/ggac467>). The code features a fully parallelized implementation, encompassing mesh loading, solution stages, output generation, and visualization. Leveraging the Portable, Extensible Toolkit for Scientific Computation (PETSc), Tandem benefits from advanced solvers, preconditioners, and time integration schemes, ensuring high performance and flexibility for complex simulations.

4.4.1 Flagship code/Mini-app(s) initial status

At the start of the project, Tandem² was mostly used as a CPU-based code, and PETSc's support for GPU data structures was not extensively tested. Initial tests on GPUs were conducted on a local server of two NVIDIA GeForce RTX 3090 GPUs, based on a simple 3D test scenario, and suggested that the GPU version could be already effective. Yet, the GPU backend was neither tested on a larger number of GPUs nor AMD GPUs, no scalability studies were conducted, and no systematic verification of the obtained results was performed. This was primarily due to the code being developed only recently and the lack of access to large GPU clusters until the ChEESE-2P project began.

The following parts will demonstrate that we now have a much better understanding and confidence in the GPU backend, its potential, and its limits.

4.4.2 Porting

Tandem benefits from using PETSc for its core structure. That means it naturally integrates current optimizations of PETSc for CPU and GPU architectures and benefits from its new developments. Different programming models, such as CUDA (for NVIDIA GPUs), HIP (for AMD GPUs), or Kokkos (for NVIDIA, AMD, or Intel GPUs) are available and can be used based on the available architectures by simply changing at runtime the `-vec_type` and `-mat_type` PETSc options.

² <https://github.com/TEAR-ERC/tandem>, version 1.0

Our Tandem spack (<https://github.com/spack/spack>) package (integrated into spack since v0.21.0) allows a straightforward installation of Tandem dependencies for the NVIDIA and AMD GPUs. We note nevertheless that on LUMI-G, tandem installed with spack will fall into a *hipSPARSE*, an issue documented on PETSc (<https://gitlab.com/petsc/petsc/-/issues/1529>), which has been fixed starting from ROCm 6.0.2. Unfortunately, the spack module based on the 23.09 programming environment has only ROCm 5.6.1. Therefore, PETSc and Tandem should be installed manually for LUMI-G. Such Tandem, based on a PETSc compiled manually based on the ROCm 6.0.3, runs successfully.

Early tests have identified that multigrid on GPU yields different convergence histories to CPUs. This issue was fixed by updating the default *vectype* property of matrix and vector objects of PETSC to a type adapted to the implementation. With this change, the Tandem mini app (the static mini app which is equivalent to execute one time step and which incapsulate the most computationally intensive part of Tandem) now yields a similar convergence history on GPU as on CPUs using both CUDA and HIP data structures.

The LUMI-G documentation suggests using a launch script to bind each CPU to the correct GPU (see *select_gpu* in <https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/lumig-job/>). This is a usual approach, e.g., on Leonardo. For unexpected reasons, this does not work properly with Tandem (on LUMI-G), Tandem then crashes depending on the number of nodes used (or GPU per node). We realized that running directly Tandem without this launch script gives good performance and permits us to avoid these unexpected behaviours.

The results from the CUDA GPU backend have been verified through verification tests and are like the CPU backend, including p- and h-convergence behaviour (Fig. 4-4.1).

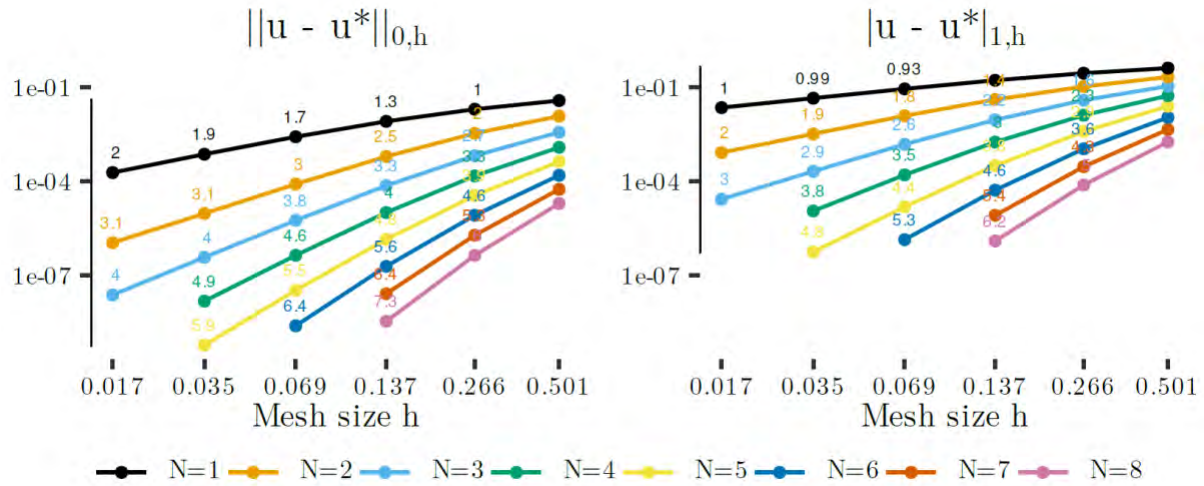


Fig. 4-4.1: Results of the spherical hole convergence tests using the CUDA backend of PETSc. This test verifies the high-order convergence of the elasticity solver using curvilinear grids. The results are identical to the one in Uphoff et al. (2022), based on the CPU backend. A few points are missing and correspond with simulations that were not fitting on the memory of the 2 NVIDIA GeForce RTX 3090 of our local cluster.

Tandem with the CUDA backend uses an unexpectedly large amount of memory, compared with the HIP backend. That is, 32 GPUs are required on Leonardo to run a relatively lightweight setup (85000 cells, p4), while 3 Graphics Compute Dies are needed on LUMI-G. Discussions³ are ongoing with PETSc developers to circumvent this problem.

Finally, we tested the strong scalability of the HIP backend on LUMI-G and compared it with the strong scalability of tandem on LUMI-C (see Fig. 4-4.2). The test case is a lightweight setup of the audit, with a mesh size of 85492 elements, and was run with p4. One LUMI-G node (4 AMD MI250x GPUs, each including two Graphics Compute Dies (GCDs)) has relatively similar performance as one LUMI-C node (2x AMD EPYC 7763, representing in total 128 cores).

³ <https://gitlab.com/petsc/petsc/-/issues/1662>

Tandem mini-app, strong scaling, P4, 85492 elements on LUMI-C and LUMI-G

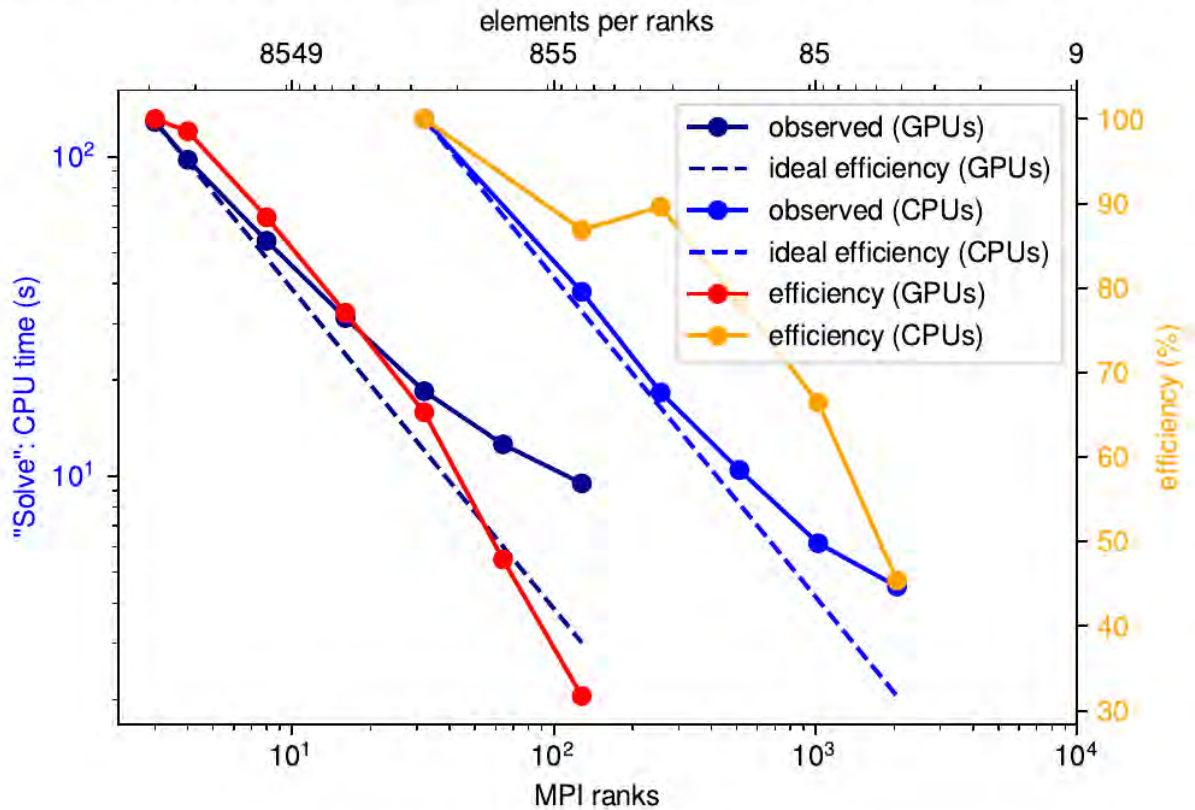


Fig. 4-4.2: strong scaling of the solve step of Tandem mini-app. The red and dark blue curves are the GPU results on LUMI-G. One MPI rank represents in this case one Graphics Compute Dies. One full node of LUMI-G is 8 MPI-ranks. The blue and orange curves are the results for LUMI-C. One full node has 128 MPI ranks.

4.4.3 Performance Portability campaign

We analysed the performance of our application in terms of TFLOP/s on LUMI-G (AMD GPUs), Leonardo (NVIDIA GPUs), and LUMI-C (CPUs) with an increasing number of devices or cores (Tab. 4-4.1). While TFLOP/s improve with the number of GPUs or CPUs, they remain far below the theoretical maximum throughput of the clusters and do not scale as efficiently as anticipated. As expected, a significant portion of the FLOP/s is offloaded to the GPUs, but this share diminishes as the number of GPUs increases:

	nodes	ranks/GPUs per node	TFLOP/s (CPU)	TFLOP/s (GPUs)	share of GPU TFLOP/s (%)
GPU (AMD)	1	3	0.013	0.132	91.3
GPU (AMD)	1	4	0.017	0.169	91.0
GPU (AMD)	1	8	0.032	0.242	88.3
GPU (AMD)	2	8	0.052	0.308	85.4
GPU (AMD)	4	8	0.090	0.326	78.4
GPU (AMD)	8	8	0.135	0.401	74.9
GPU (AMD)	16	8	0.177	0.491	73.6
GPU (NVIDIA)	16	4	0.105	1.055	91.0
CPU	1	32	0.014	0	0
CPU	1	64	0.017	0	0
CPU	1	128	0.048	0	0
CPU	2	128	0.104	0	0
CPU	4	128	0.190	0	0
CPU	8	128	0.327	0	0
CPU	16	128	0.463	0	0

Tab. 4-4.1: Total TFLOP/s, Solve stage of Tandem on LUMI-C and LUMI-G

To better understand differences in scalability between the CPU and GPU runs, we examine the percentage of time spent in the most time-intensive phases of the *Solve* stage in Tandem (Tab. 4-4.2). On LUMI-C, we observe that *MatSolve* accounts for up to 56% of the runtime with 32 cores. However, as the CPU count increases, the runtime share of *MatSolve* decreases, while phases such as *VecScatterEnd*, *MatMultAdd*, and *VecTDot* begin to dominate.

On GPUs, the largest portion of the runtime is spent on *MatMult*, accounting for up to 86% of the total time when using three devices, while *MatSolve* only consumes 6% of the runtime in this configuration. However, as the number of GPUs increases, the relative time spent in *MatSolve* grows, reaching up to 14%. Additionally, other phases such as *VecScatterEnd*, *MatMultAdd*, and *VecTDot* progressively take on a more significant share of the runtime. Results are consistent with both CUDA and HIP. We note that these phases, including *MatMult*, *MatSolve*, *VecScatterEnd*, *MatMultAdd*, and *VecTDot*, are components of PETSc rather than Tandem itself.

	ranks /GPUs		PCAp ply	MatSol ve	Mat Mult	VecScat terEnd	MatMult Add	VecTDot	Mat Residual	VecNorm	VecScatter Begin	MatMult Transpose
	node s	per node										
GPU (AMD)	1	3	91	6	89	0	0	2	8	0	0	1
GPU (AMD)	1	4	91	6	89	1	0	3	9	0	0	1
GPU (AMD)	1	8	91	6	87	1	1	2	9	0	1	2
GPU (AMD)	2	8	92	7	81	2	2	4	9	0	1	2
GPU (AMD)	4	8	94	10	72	6	3	5	9	1	2	3
GPU (AMD)	8	8	95	13	61	10	5	7	9	1	3	3
GPU (AMD)	16	8	96	14	53	16	7	7	9	2	5	4
GPU (NVIDIA)	16	4	93	6	79	13	3	4	10	1	2	2
CPU	1	32	98	56	33	4	2	2	6	1	0	3
CPU	1	64	98	64	24	5	3	3	5	1	0	3
CPU	1	128	95	54	30	6	3	4	5	3	1	3
CPU	2	128	98	55	30	8	3	4	5	2	2	3
CPU	4	128	97	49	28	12	6	5	5	2	2	3
CPU	8	128	97	40	24	17	11	6	4	3	3	2
CPU	16	128	95	27	20	23	16	6	3	6	3	1

Tab. 4-4.2: Percentage time in the most time-consuming phases of the Solve stage of Tandem on LUMI-C, LUMI-G, Leonardo

4.4.4 Code Audit driven improvements

The first code audit, conducted on the CPU-based server LUMI-C, and based on the mini app, has identified that the strong scaling is severely limited by both growing load imbalance and rapidly growing MPI point-to-point communication. The audit was challenged by the prolific use of *MPI_Iprobe*, *MPI_Test* & *MPI_Testall* by PETSc, making Score-P tracing impractical. Also, Solve based on PETSc and related call structure were unfortunately not distinguished in measurement. Some unexpected behaviours were identified, such as one specific rank that sends and receives much more data than others and takes more than twice as much compute time with 8192 MPI ranks. Unfortunately, clear recommendations to solve this problem could not be formulated, without further instrumentation of the code. Manual annotation of code with Score-P marker routines has been started recently and could offer some avenues for improving the strong scalability of tandem in the continuation of the project. In summary, no code-audit-driven improvements can be documented at this stage, but ongoing efforts may enable such improvements.

4.4.5 Other improvements

Tandem requires solving many time steps, which in combination with the identified strong-scalability limits require long run time. Most HPC systems limit run time to a few days and advantage large node counts jobs over long run time jobs. Therefore, it is essential to be able to chunk simulations into several jobs, using time checkpointing. This feature has been implemented during the project.

Tandem time to solution benefits from precomputing Green's functions in a preliminary stage and combining the Green function results efficiently in a later stage. The functionality to checkpoint these Green's functions has been developed, also allowing to chunk production size simulations into several jobs.

Finally, optimization in the Rate and State friction routines, inverting for the logarithm of the slip rate, instead of directly for the slip rate has been implemented and allows 30% speed up of the time-stepping loop. Other improvements, such as weighted mesh partitioning, penalizing fault elements, and improved multi-grid preconditioner are in development and are expected to further reduce the time to solution.

4.4.6 Exascale ambition

Tandem is evolving rapidly to address the challenges and opportunities posed by pre-Exascale and Exascale computing platforms. Initially designed as a CPU-based code, Tandem has been extended to support GPU architectures, leveraging PETSc's evolving GPU backends (CUDA, HIP, and Kokkos). These developments enable the code to exploit heterogeneous systems and adapt to modern exascale infrastructures, such as those in Euro-HPC, while maintaining consistency in convergence history across CPU and GPU platforms.

Our new analysis on LUMI-G and Leonardo has identified the current scalability behaviour of Tandem on AMD and NVIDIA clusters, as well as areas for improvement. For example, the HIP backend demonstrates better memory efficiency than the CUDA backend, reducing the number of GPUs required for large simulations. Ongoing collaborations with PETSc developers aim to mitigate memory usage issues, identify bottlenecks in FLOPS throughput, and further enhance the scalability and resource efficiency of Tandem on GPUs. In conjunction with these efforts, instrumentation with tools like Score-P and manual code annotations is underway to diagnose and optimize performance bottlenecks.

4.5 xSHELLS

XSHELLS effort in WP2 is spent in the two code audit steps. Due to those constraints there are no activities to report in D2.2.

4.6 HySEA

Tsunami-HySEA is a high-performance numerical model for simulating tsunamis generated by earthquakes. The model integrates all three phases of an earthquake-generated tsunami—seafloor deformation, tsunami propagation, and coastal inundation—into a single cohesive code. Tsunami-HySEA supports a range of advanced features, including nested grids, multiple seafloor deformations at any time, time series analysis, compressed NetCDF output, asynchronous file writing, multiple performance metrics, dynamic deformation, and variable friction. These capabilities make it a powerful tool for detailed tsunami modeling and analysis.

4.6.1 Flagship code/Mini-app(s) initial status

The Tsunami-HySEA code had already passed through two performance audits that were carried out by the POP Centre of Excellence in HPC and within the framework of the ChEESE project. This means that the Tsunami-HySEA European flagship code for the simulation of earthquake generated tsunamis, was already very efficient at the start of ChEESE-2P, as demonstrated by the first audit performed now during this project, where Tsunami-HySEA achieved good strong scaling and only some minor issues were encountered (more details in section 4.6.4).

The Tsunami-HySEA mini app consists of a reduced version of the flagship code without some functionalities, such as the capability of dealing with nested grids and performing multi-GPU simulations. At the beginning of ChEESE-2P, we already had an initial CUDA version of the mini app. In the first months of the project, we improved this initial version by adding new functionalities, such as performing Okada deformations and reading the input grid from a NetCDF file. At M4 (April 2023) we created a GitHub repository storing the first publicly available HySEA mini app, version 1.1.1, implemented in CUDA (<https://github.com/edanya-uma/TsunamiHySEA>).

4.6.2 Porting

In March 2023, we started a collaboration with Intel Corporation to port our Tsunami-HySEA CUDA mini-app version 1.1.1 to SYCL. The SYCL implementation of the mini app was finished in June, and a public GitHub repository for this version was created in September 2023 (<https://github.com/edanya-uma/TsunamiHySEA-SYCL>).

Both CUDA and SYCL versions of the mini app have the same functionalities. In the last months we have added more features to both versions of the mini app. Specifically, we have implemented the saving of the arrival times in the output NetCDF file, the possibility of performing multiple Okada deformations at selected time instants, and an algorithm to crop the obtained deformations by means of a relative or absolute threshold value. However, the performance portability campaign was carried out using the current version 1.1.1 of the mini-app available in GitHub, which does not contain these additions.

The SYCL version of the mini app has been successfully tested on Intel CPUs, and on the following GPUs: NVIDIA H100, Intel Data Center GPU Max 1100, and Intel Data Center GPU Max 1550.

4.6.3 Performance Portability campaign

The test case that will be used in this section consists of a tsunami simulation in the Mediterranean Sea using a mesh of 10 of millions of cells with a spatial resolution of 30 arc-sec. The simulation time is four hours.

Scaling

The scaling of the SYCL version of the mini app has been obtained on an Intel Emerald Rapids 8952 processor, having 64 cores per socket, 2 sockets and 2 threads per core. We have considered up to 256 cores, where hyperthreading has been used for the 256 cores run. The code has been compiled with the Intel oneAPI Toolkit 2024.2.1, the CPU was targeted with the `ONE_API_DEVICE_SELECTOR="opencl:cpu"` compiler flag, and the `taskset` command was employed to select the number of cores.

In Fig. 4-6.1, the table and the chart depict the runtimes in seconds of the different executions and the obtained scalability, respectively.

A deeper analysis shows that approximately 85 % of the runtime is employed on the CPU, and 15 % on memory bandwidth. Thus, we can conclude that the SYCL Tsunami-HySEA mini-app presents a low dependency on the memory bandwidth, and a high memory bandwidth will not provide a better speedup in this case.

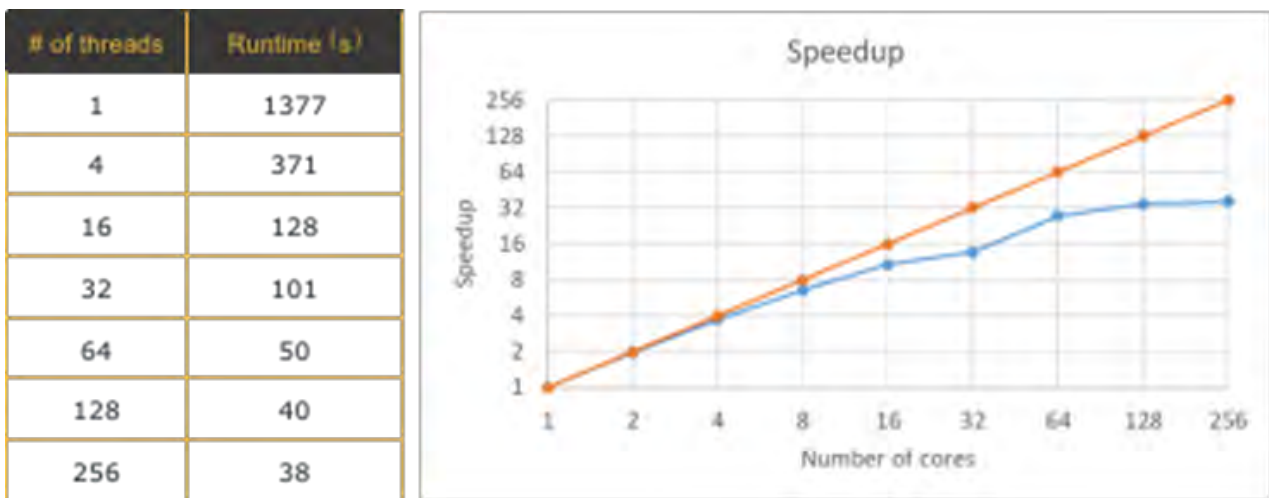


Fig. 4-6.1: Runtime and Speed-up of SYCL Tsunami-HySea version.

Runtimes

Both SYCL and CUDA versions of the mini app have been tested on multiple platforms. Specifically, the SYCL mini app has been preliminarily executed on three different GPUs, namely the NVIDIA H100, the Intel Data Center GPU Max 1550 and the Intel Data Center GPU Max 1100, using the Intel

DPC++ compiler based on LLVM (Intel LLVM). For the complete performance portability analysis, we added two extra platforms: Leonardo Booster, featuring NVIDIA A100 GPUs, and the Thea cluster from NVIDIA, based on the recent GH200 “superchip”. Furthermore, we extended our analysis to consider three distinct SYCL compiler implementations: the already mentioned Intel LLVM, the compiler provided together with Intel’s oneAPI toolkit, and the AdaptiveCpp compiler. Where possible we repeated runs on already tested platforms using another compiler to increase data coverage. Tab. 4-6.1 shows runtimes in seconds for the various tested configurations and the corresponding application efficiencies. The latter are computed as ratios between the best runtime on each platform and the runtime for the specific configuration.

	CUDA		oneAPI		Intel LLVM		AdaptiveCpp	
GPU	Runtime	Efficiency	Runtime	Efficiency	Runtime	Efficiency	Runtime	Efficiency
NVIDIA A100	102.51	0.97	119.70	0.83	99.30	1.00	101.39	0.98
NVIDIA H100	74.70	1.00	N.A.	N.A.	76.60	0.98	N.A.	N.A.
NVIDIA GH200	53.27	0.98	N.A.	N.A.	N.A.	N.A.	52.14	1.00
Intel Max 1100	N.A.	N.A.	215.80	0.97	209.00	1.00	N.A.	N.A.
Intel Max 1550	N.A.	N.A.	119.00	0.45	53.10	1.00	N.A.	N.A.

Tab. 4-6.1: Runtime and Efficiency for different configuration.

Plotting the efficiencies reported above in a cascade plot we get the Fig. 4-6.2. The left plot features CUDA together with all the distinct SYCL implementations. We can see that overall Intel LLVM comes out as the most efficient and portable implementation, both at the SYCL level and globally (i.e., comparing with CUDA). The other SYCL implementations are either not tested as extensively, or not as efficient as Intel LLVM. It should be noted that for the moment no SYCL implementation seems to be portable across the full set of selected platforms. This lack of portability is also the reason of the null values of performance portability shown in the small chart on the right of the cascade plot.

On the other hand, the right plot provides a global overview on the efficiency of the SYCL version independently of the specific compiler. In this case the efficiency corresponds to the best value obtained by the SYCL mini app on a specific platform (possibly corresponding to different compilers on different platforms). From the plot we can see that allowing flexibility on the compiler, the SYCL version is indeed fully portable (see the performance portability

chart on the right of the plot), and it also reaches very high levels of efficiency on all the tested platforms (the minimum being the 0.98 efficiency on the H100).

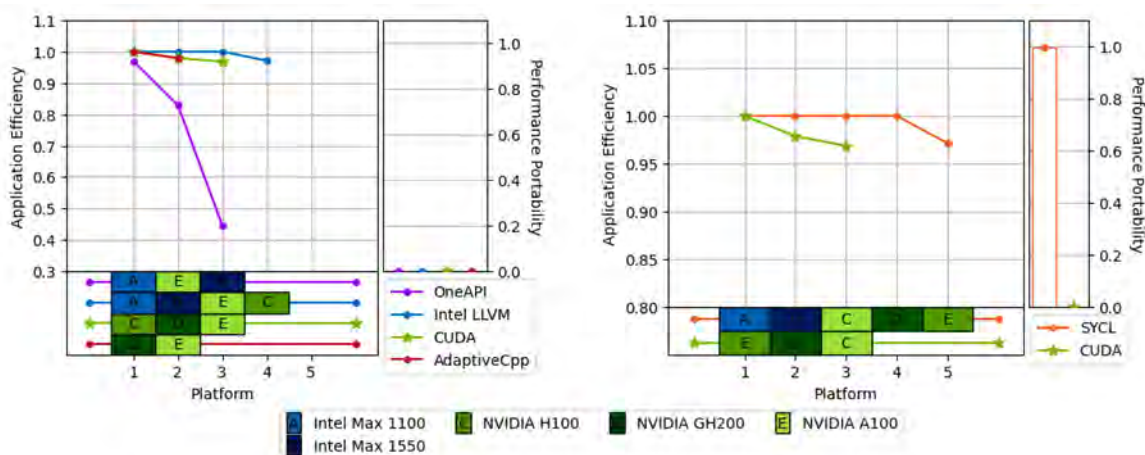


Fig 4-6.2: Performance Portability chart

Tab. 4-6.2 shows PP values computed on subsets of the full set of tested platforms. For a given number of platforms N, the max value is computed using the N best efficiencies, while the min value using the N worst efficiencies. For any other subset of N platforms, the application will have a value of PP metric comprised within the minimum and the maximum. The performance portability level of the various application versions is relatively

stable, except for the oneAPI toolkit case, where we have a larger variability (a larger range), as we can also see from the sharp drop in the efficiency plot. This is probably due to some inefficiencies in the software stack on nodes where we performed tested Intel GPUs.

	1 Platform		2 Platforms		3 Platforms		4 Platforms		Full set (5)	
Version	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min
SYCL	1.0	0.97	1.0	0.99	1.0	0.99	1.0	0.99	0.99	0.99
Intel LLVM	1.0	0.97	1.0	0.99	1.0	0.99	0.99	0.99	N.A.	N.A.
CUDA	1.0	0.97	0.99	0.97	0.98	0.98	N.A.	N.A.	N.A.	N.A.
oneAPI	0.97	0.45	0.89	0.58	0.70	0.70	N.A.	N.A.	N.A.	N.A.
AdaptiveCpp	1.0	0.97	0.99	0.99	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

Tab. 4-6.2: Performance Portability values

4.6.4 Code Audit driven improvements

Tsunami-HySEA performed well in the first code audit carried out in the ChEESE-2P project. We used a test case which simulated the tsunami caused by the 2011 Tohoku earthquake on a 30 arc-sec grid, achieving a speedup of 57.4 using 64 GPUs on the Leonardo supercomputer (with respect to 1 GPU). Only minor issues were reported by the audit: some CUDA kernels with low load balancing numbers (the kernel that processes the communication edges and the two kernels that update the state of a volume at each iteration of the simulation). All these kernels have been modified by compacting instructions to try to improve their load balancing. The new code with these modifications is finished but the resulting scaling has not been evaluated yet.

4.6.5 Other improvements

No other efficiency improvements besides those described in this document have been implemented in the Tsunami-HySEA flagship code or the mini app. However, a couple of new functionalities have been added to the flagship code after the audit. An updated algorithm to compute the triangular faults (Nikkhoo-Walter), and the same algorithm to crop the obtained deformations that was implemented in the mini app.

4.6.6 Exascale ambition

The aim of ChEESE-2P is to drive toward exascale computing Solid Earth (SE) applications and workflows, which represents a transformative ambition in computational science in SE. The two-fold objective is to push both capacity and capability computing to unprecedented levels. Within this context, codes, applications and workflows including tsunami simulations in ChEESE-2P look to achieve both capacity and capability computing, addressing large high-resolution simulation problems as well as applications requiring a huge number of numerical simulations, for example in probabilistic approaches, that must post-process a huge amount of data. Multi-source tsunami simulations within PD4 and the associated SC4.1 fits into the capability computing context, while PD3 and SC3.1 aims at capacity computations.

4.7 FALL3D

FALL3D is a Eulerian model for simulating the atmospheric transport and ground deposition of particles, radionuclides, and aerosols, including

volcanic tephra, ash, and gases (Folch et al., 2020 <https://doi.org/10.5194/gmd-13-1431-2020>). The model employs a high-resolution central-upwind scheme based on finite volumes to solve advection-diffusion-sedimentation (ADS) equations on a structured grid. This approach minimizes numerical diffusion, even in the presence of sharp concentration gradients and discontinuities, ensuring accurate and reliable simulations of complex atmospheric processes.

4.7.1 Flagship code/Mini-app(s) initial status

The initial status of the FALL3D suite (<https://gitlab.com/fall3d-suite>) included 3 primary code repositories:

1. A full CPU version 8.3,
2. A GPU-accelerated version based on CPU version 8.2 and,
3. A simplified mini app with both CPU and GPU variants.

While the CPU versions provided flexibility and portability, the GPU versions (inherited from ChEESE original project) were specifically tailored and optimised for NVIDIA hardware. These GPU versions delivered notable performance but were constrained in terms of hardware compatibility (e.g. could not run on AMD or Intel accelerators). On the other hand, the fragmentation in the suite, with different codes and repositories for CPU/GPU/Mini app(s), was not optimal in terms of user experience, code maintenance, and portability, as detailed in the following sections.

FALL3D full app

The full app was originally split in 2 distinct repositories: one targeting CPUs (version 8.3) and another accelerated using OpenACC directives, specifically targeting NVIDIA GPUs (derived from CPU version 8.2). The CPU version 8.3 included significant enhancements over its predecessor (e.g. improved management of memory to overcome code restrictions when reading large meteorological files, additional meteorological models and some bug fixes) so that significant porting effort was required to upgrade GPU version 8.2 to match its CPU counterpart. Added to this, the accelerated code was constrained to NVIDIA devices and could not be compiled in architectures from other vendors. Such scenario presented several issues:

- Users experience complexity: Users were required to choose between two repositories with varying feature sets and development levels, complicating adoption and usability.

- Maintenance overhead: Independent maintenance, documentation, and development were required for each repository, increasing the workload for developers.
- Portability limitations: Dependency on a specific accelerator vendor restricted the GPU code deployment on certain systems (e.g. LUMI).

FALL3D Mini app

Similarly, the mini app with GPU support was compatible just with NVIDIA hardware, mirroring the challenges faced by the full app in terms of portability, maintenance, and user experience complexity. In addition, the code inherited from ChEESE faced additional issues. For example, this code corresponds to a very early stage in the development process for GPU porting. Consequently, the mini app lacked some important optimizations included in the GPU version of the full app and the divergence between the full-app and the mini-app codes were substantial. All these issues led to duplicate development efforts, error-prone code and was also problematic in terms of co-design, limiting future activities in WP2.

Unified FALL3D version 9.0

After evaluating various porting options including standard Fortran parallelization, OpenACC, OpenMP offloading, and frameworks such as OpenCL, Raja, and Kokkos, a decision was made to consolidate developments into a unified code version 9.0. Given the pre-existing OpenACC accelerated version (8.2) this process was not a traditional porting effort but, nonetheless, significant efforts were required to refactor in terms of portability (see Section 4.7.3). The present version 9.0 (<https://gitlab.com/fall3d-suite/fall3d/-/releases/9.0> released in October 2024) supports both CPU and GPU executions within a single codebase and runs across heterogeneous architectures, including AMD and NVIDIA devices on target EuroHPC systems (LUMI, Leonardo, and MareNostrum 5). OpenACC was selected due to its compatibility with Fortran, support for multiple hardware vendors and the relatively low entry barrier. To support developers unfamiliar with OpenACC, detailed comments were added to the code in order to guide developers through data management areas and clearly annotating GPU and CPU regions.

Currently, version 9.0 accelerates about 40% of the code standard workflow while retaining all new functionalities introduced in version 8.3. Initial performance tests, particularly comparing the results against the still much more specialised GPU version 8.2, suggest that there is still room for

further optimizations, but the code is now operational across all target systems. Future releases are expected to achieve further performance gains, with the target of achieving times-to-solution like 8.2.

4.7.2 Porting

As all the codes are OpenACC based, across the developed versions, no real porting was done (in the framework of WP2).

4.7.3 Performance Portability campaign

As outlined in section 4.7.1, significant efforts have been made to enhance both the performance and portability of the codebase. Extensive testing across multiple architectures and compilers has confirmed robustness (exactly same results) and portability in the CPU code. However, the most significant advances have been in the accelerator domain. While the previous GPU version 8.2 was optimised for NVIDIA devices, and specially for V100 accelerators that were available in MareNostrum 4 during ChEESE project, the new unified version 9.0 can be successfully executed on both NVIDIA (V100, A100 and H100) and AMD (MI250X) accelerators.

The development of FALL3D version 9.0 and the related Performance Portability Campaign (PPC) has been possible thanks to 4 development access projects, in Tab. 4-7.1:

Reference	System	Node h.	Start	End
EHPC-DEV-2023D07-008	Leonardo	3.500	01/09/2023	31/08/2024
EHPC-DEV-2023D10-028	Lumi G	10.000	15/10/2023	15/10/2024
EHPC-DEV-2024D09-010	Leonardo	4500	27/09/2024	27/09/2025
EHPC-DEV-2024D09-012	Lumi G	4500	18/09/2024	18/09/2025

Tab. 4-7.1: Overview of the EuroHPC projects obtained by the FALL3D code

To achieve a balance between performance and portability, a strategic decision was made to consolidate into a single, versatile codebase. This approach, while potentially sacrificing some performance, offers significant

advantages in terms of maintainability, usability across diverse HPC platforms, and future adaptability. The combined MPI and OpenACC approach has been adopted to optimise both performance and portability, leveraging the strengths of each technology:

- **Scalability:** MPI provides a robust, well-established framework for parallel computing, enabling efficient scaling across multiple nodes and cores. While MPI can function independently of OpenACC, it serves as a foundational parallelization approach that supports diverse computational strategies.
- **Acceleration:** OpenACC offers an accessible and effective way of offloading computationally intensive code segments to accelerators without requiring complex, low-level programming or extensive code refactoring.
- **Portability:** Both MPI and OpenACC are supported on the target HPC systems, ensuring that the resulting code can be compiled across these different hardware platforms.
- **Complementary integration:** Assigning one MPI rank per GPU simplifies the code comprehension for developers and supports efficient parallel execution. MPI manages communication across different segments of the computational domain, while OpenACC handles intra-node acceleration, effectively utilising GPUs for high-intensity tasks. This combination achieves a scalable and performant solution by exploiting the strengths of both technologies.

This approach aligns with the goal of creating a portable codebase that can operate effectively on various HPC platforms.

Portability tests

To validate portability across target systems, a comprehensive evaluation of compiler and system compatibility was conducted. Tab. 4-7.2 summarises the code portability test results, providing an overview of the validated combinations.

Portability tests		System		
Compiler	Compilation	Leonardo	LUMI	MN5
GNU	Serial	■ ● ★	■ ● ★	■ ● ★
	MPI	■ ● ★	■ ● ★	■ ● ★
NVHPC	Serial	■ ● ★	N.A.	■ ● ★
	MPI	■ ● ★	N.A.	■ ● ★
	MPI + ACC	■ ● ★	N.A.	■ ● ★
CRAY	Serial	N.A.	■ ● ★	N.A.
	MPI	N.A.	■ ● ★	N.A.
	MPI + ACC	N.A.	■ ● ★	N.A.

Tab. 4-7.2: Combinations of compiler and the compilation type on different EuroHPC systems. Star symbols (★) denote the new portable version 9.0, squares (■) represent the CPU version 8.3 and circles (●) the GPU version 8.2. In any case, green symbols indicate successful compilation and testing, while red symbols indicate incompatibility. The cells with NA indicate the unavailability of the compiler in the system.

Strong Scalability tests

To evaluate the scalability of FALL3D versions 8.2 and 9.0 on MareNostrum 5, and of version 9.0 on LUMI, a series of tests were conducted using increasing resource allocations.

The strong scalability (9.0) and the performance (8.2 and 9.0) test case is based on the simulation of the 2019 Raikoke eruption (northwest Pacific Ocean) which is one of the test cases included in the FALL3D test suite (<https://gitlab.geo3bcn.csic.es/fall3d/test-suite>). The case considers a 1-hour simulation with the restart option enabled to define a non-zero initial condition based on the concentration field generated from a previous simulation of the eruption. The domain is composed of 1000×1000 cells and 100 vertical levels (100 million of grid cells in total) and 5 particle bins corresponding to tephra species (fine ash). This large problem was defined by maximising the size of the domain in such a way as to allocate the maximum available memory on a single node of Leonardo.

As shown in Fig. 4-7.1(a), time to solution results on MN5 for versions 8.2 and 9.0 indicate that, while the initial execution time with one GPU in version 9.0 is nearly ten times slower than in version 8.2, this difference decreases as resources are scaled up. The slower performance on a single GPU in version 9.0 is anticipated, as substantial parts of the code, particularly sections for meteorological data processing, above 60% of the usual code workflow, are not fully accelerated yet.

On LUMI, Fig. 4-7.1(b) reveals a different scalability pattern. An unusual trend in scaling appears until the full utilisation of a node (eight GCDs) is reached. When scaling from two to four GCDs, performance improves only marginally, with a slight time increase for four GCDs. This behaviour normalises with the allocation of a full node (eight GCDs), where a minor reduction in time is observed. The scalability becomes pronounced with 16 GCDs and shows important improvements with 32 GCDs, where execution time drops sharply. With 64 GCDs, the time to solution continues scaling well. LUMI support team suggests that data movement in and out of memory, likely due to memory management patterns, may account for the irregular scaling below a full node. Additionally, the development team suspects that the inability to assign specific GPUs and CPUs in LUMI's configuration may cause inefficient data transfers across the CPU-GPU bus, contributing to the observed performance irregularities in sub-node configurations.

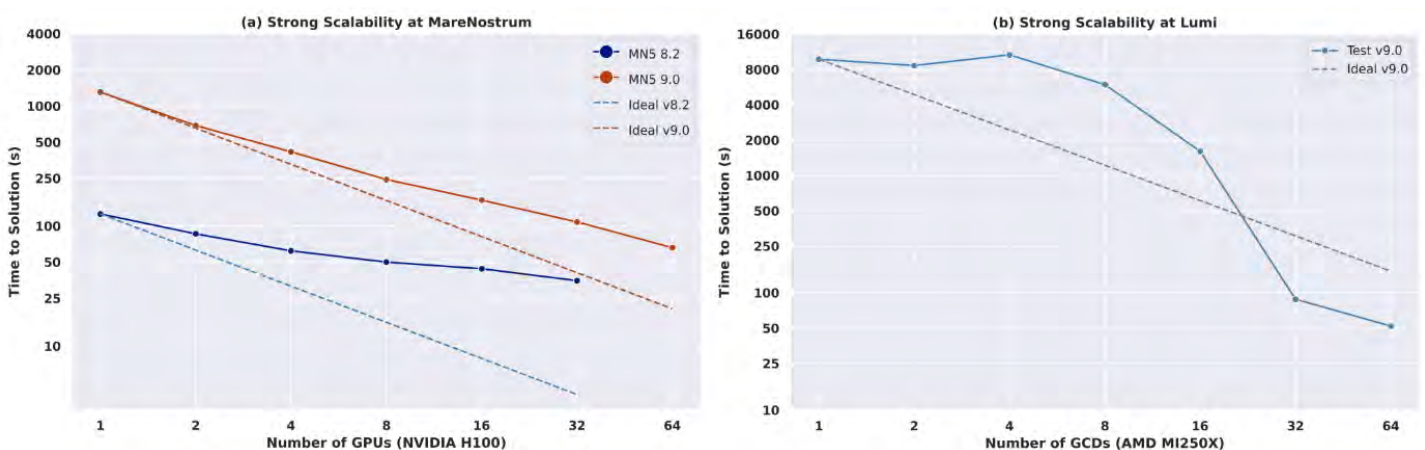


Fig. 4-7.1: (a) Strong scalability tests on MareNostrum 5 for versions 8.2 and 9.0. (b) Strong scalability test on LUMI for version 9.0.

Fig. 4-7.2 presents the parallel efficiency results for both the MN5 and LUMI platforms, showing scalability as a percentage of parallel efficiency across the three cases. On LUMI, parallel efficiency reaches up to 350% at 32 GCDs, likely due to the unaccelerated portion of the meteorological code

and the inefficient data transfers across the CPU-GPU bus discussed earlier. With increased resources, additional CPUs are leveraged for these unaccelerated meteorological data processing tasks. Furthermore, as GPU allocation increases, the communication bandwidth between CPU and GPU is improved, effectively reducing the global computational load and communication overhead. This effect could be considered beneficial for the data transfers between CPU and GPU needed by the unaccelerated meteorological portion.

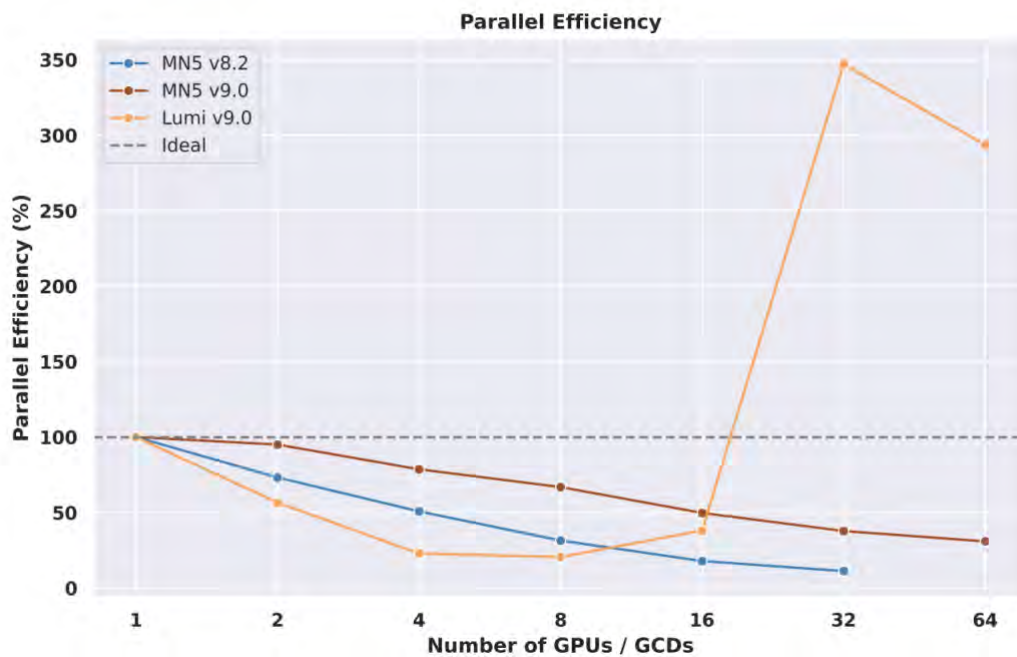


Fig. 4-7.2: Parallel efficiency results for version 8.2 on MareNostrum 5 and version 9.0 on both MareNostrum 5 and LUMI.

In MareNostrum 5, version 9.0 shows better scaling compared to version 8.2, even though it begins with a slower baseline. This scaling benefit likely arises from the additional CPU resources available in the scaling resource increase, aiding in the processing of meteorological data, like the trend observed on LUMI. As optimization of version 9.0 progresses, ongoing work aims to fully accelerate the remaining code sections and further address current inefficiencies. Meanwhile, the tests conducted on MareNostrum 5, Leonardo, and LUMI have validated correct operational behaviour across platforms, providing a good base for the next stages of improvement.

Performance tests

Performance tests were conducted to evaluate and compare the computational efficiency of the FALL3D code versions across target HPC platforms using 2 nodes (8 GPUs) in Leonardo and MareNostrum 5, and 1 node (8 AMD GCDs) in LUMI, using the simulation detailed in section 4.7.3. These tests focused on identifying performance improvements, challenges, and the impact of recent code optimizations.

This case was tested with the following FALL3D code versions:

- Version 8.2 GPU ACC+MPI: Original GPU accelerated version using OpenACC and MPI domain decomposition, tailored to NVIDIA GPUs.
- Version 9.0 GPU ACC+MPI: The latest portable version, combining GPU acceleration with OpenACC and MPI domain decomposition.

The Tab. 4-7.3 summarises the computational times recorded for each version across the different clusters:

FALL3D compilation	Cluster	Decomposition	Computing time
V 8.2 GPU ACC+MPI	Leonardo	X=2, Y=2, Z=2	39"
	MareNostrum 5	X=2, Y=2, Z=2	50"
	Lumi	N.A.	
V 9.0 GPU ACC+MPI	Leonardo	X=2, Y=2, Z=2	486"
	MareNostrum 5	X=2, Y=2, Z=2	244"
	Lumi	X=2, Y=2, Z=2	1232"

Tab. 4-7.3: Performance test results for the two GPU versions of the code, 8.2 and 9.0.

The results highlight the following:

- Version 8.2 GPU ACC+MPI consistently outperforms version 9.0 on NVIDIA hardware (achieving a total computation time of 39 and 50 seconds on Leonardo and MareNostrum 5 respectively), demonstrating its much higher efficiency on specific hardware.
- Version 9.0 GPU ACC+MPI is portable but still requires further optimization to enhance GPU performance across different architectures.

Current tests demonstrate that version 9.0 is not yet achieving expected performance levels for accelerated code due to approximately 60% of the workflow remaining without GPU acceleration. Consequently, unnecessary

data transfers between the host and the accelerator are still occurring. While the exceptional performance of version 8.2 on Leonardo serves as a benchmark, achieving similar performance may be challenging due to the specific characteristics of NVIDIA hardware and its proprietary API options. Nonetheless, efforts are focused on approaching this benchmark as closely as possible during the project.

Performance Portability Metrics

To quantify the enhancements from the PPC, the performance portability (PP) metric proposed by S. John Pennycook et al. ([10.1109/MCSE.2021.3097276](https://arxiv.org/abs/10.1109/MCSE.2021.3097276)) has been employed. This PP metric combines efficiency and portability, rated from 0 to 1. Efficiency is calculated per platform as the ratio between the theoretical optimal runtime and the actual runtime achieved, with 1 representing ideal performance. To aggregate this measure, the harmonic mean of efficiencies across platforms is used, which emphasises the impact of inefficiencies on any single platform by significantly lowering the overall score.

FALL3D	Cluster	Domain	Time	Efficiency	PP metric
V 8.2	Leonardo	X=4, Y=2, Z=2	66"	0.67	0
	MareNostrum 5	X=4, Y=2, Z=2	44"	1	
	LUMI	N.A.			
V 9.0	Leonardo	X=4, Y=2, Z=2	567"	0.08	0.057
	LUMI	X=4, Y=2, Z=2	1605"	0.92	
	MareNostrum 5	X=4, Y=2, Z=2	159"	0.28	

Tab. 4-7.4: Performance Portability metric and platform efficiency of the target clusters for the two GPU version of the code.

The overall results obtained for each version are:

- Version 8.2: PP=0, indicating non portability as it only can run on two of the three target platforms. However, this version serves as a baseline in means of wall time for measuring PP metric as it is the faster execution for the same problem across all versions and platforms.
- Version 9.0: PP=0.057, reflecting compatibility with all target platforms but with lower performance efficiency.

The radar plot in Fig. 4-7.3 visualises the efficiency individual score across the target platforms, with version 8.3 in red and version 9.0 in blue. This visualisation will be used to assess future code releases, tracking performance portability progress over time.

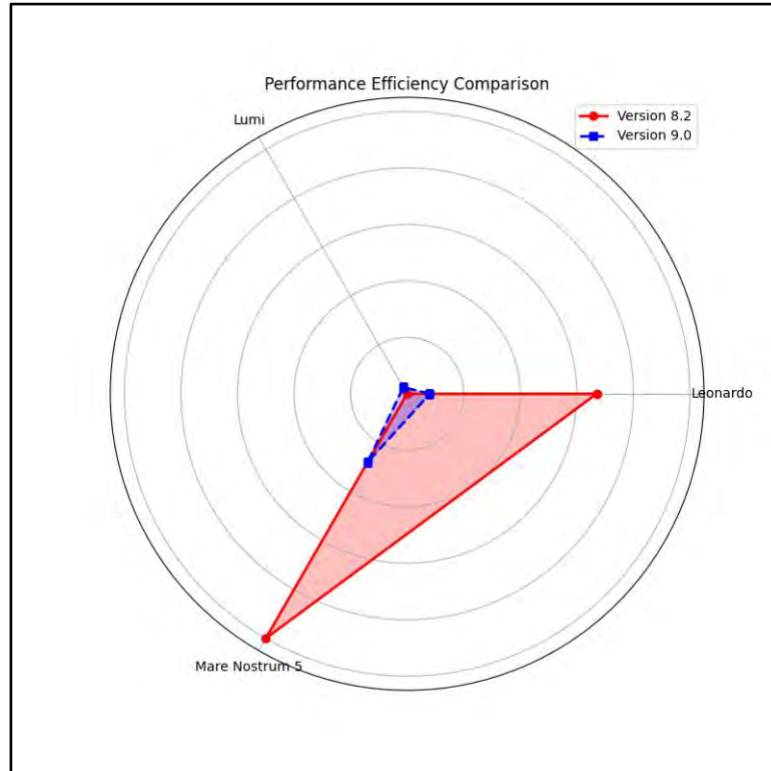


Fig. 4-7.3: Comparative diagram showing the results of the Performance Portability (PP) metric for different platforms: LUMI, MareNostrum 5 and Leonardo. This metric ranges from 0 (worst) to 1 (best) and quantifies both performance and portability.

4.7.4 Code Audit driven improvements

To identify performance issues and optimization opportunities, a code audit was conducted by POP3 teams, focusing on GPU version 8.2 (see deliverable D2.1 for details). Although version 8.2 differs from the portable version 9.0, the audit's insights proved to be valuable as both versions employ OpenACC. By addressing communication patterns and kernel inefficiencies identified in version 8.2, the development of the portable implementation avoided similar problems. Additionally, specific optimization recommendations from the audit were incorporated into the portable version 9.0.

Unnecessary CUDA-aware communications

One significant performance issue identified was inefficient CUDA-aware communication during the code's initialization stage. Fig. 4-7.4 shows the runtimes for MPI, OpenACC, and executed kernels, highlighting the

execution of small kernels that transmit data from the GPU using CUDA-aware MPI. This process was inefficient because data initially residing on the CPU was first transferred to the GPU, only to be communicated back via GPU. The MPI domain decomposition introduces dependencies between data ranges, but transferring data between the host and GPU introduced unnecessary delays.

In contrast, in version 9.0, a new function using device aware MPI has been implemented. The CPU function is used if data resides on the CPU, and the device aware function is called if data is on the GPU, removing superfluous data transfers. Depending on future code requirements and the code acceleration progress, this approach may evolve into a unified function or remain as a dual path solution.

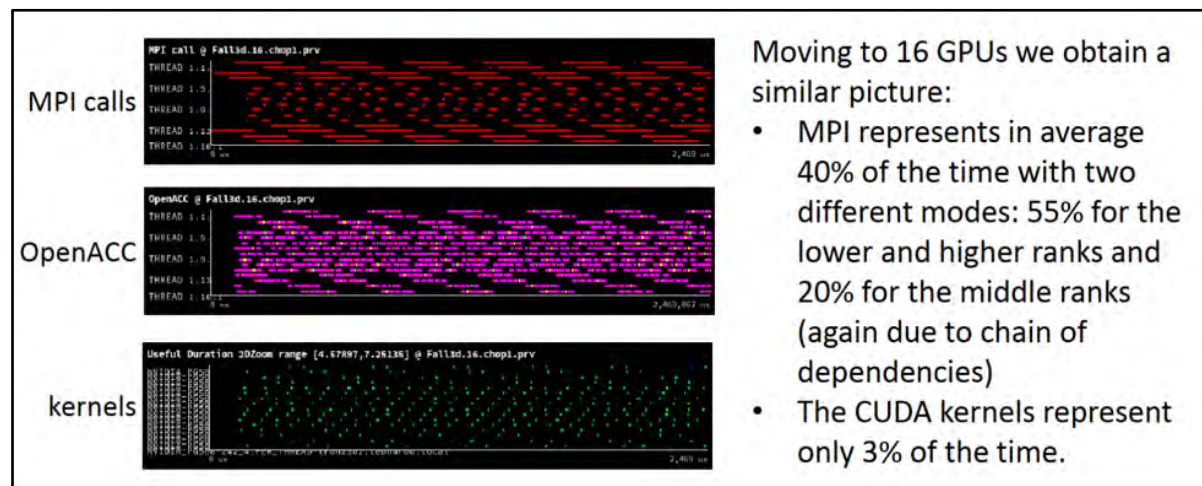


Fig. 4-7.4: OpenACC and CUDA runtime alongside MPI call for the CUDA aware communications.

MPI load

For the code audit test, performance degradation was observed when the domain was subdivided into more than 8 subdomains, primarily due to increased data transfer overhead associated with Z-axis division. To address this, critical code sections within frequently invoked functions were restructured to reduce MPI communication load. Tab. 4-7.5 shows audit data collected for analysis. Although a precise performance metric is not yet available, preliminary estimates indicate a roughly 50% reduction in MPI communication overhead into these functions. A comprehensive performance evaluation will follow in the next POP3 audit to quantify these optimizations.

MPI routine	Metrics (MPI process)	Domain decomp. 2-2-1	Domain decomp. 4-2-2
MPI_Wait	no. of calls	256	448
	time (μ s)	54	42
MPI_Irecv	no. of calls	128	224
	time (μ s)	1.7	1.7
MPI_Isend	no. of calls	128	224
	time (μ s)	4.1	3.2
MPI_Allreduce	no. of calls	4	4
	time (μ s)	141	202.1

Tab. 4-7.5: Statistics of MPI routines called by the code, obtained from the code audit conducted by the POP3 team.

Kernel Optimization

The code audit identified 40 distinct kernels distributed across 7 routines as can be seen in Fig. 4-7.5 for an execution with 4 GPUs, where each kernel is represented with a different colour. This configuration introduces a small overhead, as each kernel must be created with OpenACC and dispatched to the device for execution. Among these, some kernels were pinpointed from the POP3 audit for review. In FALL3D, such distribution of kernels is difficult to avoid, as the primary performance constraint on GPUs appears from the data movement of the computational volume between the CPU and GPU. Unlike applications with a central, computationally intensive function, FALL3D initiates the computational volume on the GPU and dispatches individual calculations and communications directly from the device.



Fig. 4-7.5: Distinct kernels called by code in an execution with 4 GPUs, represented in different colours.

To mitigate the performance impact, 2 primary actions were taken:

- **Function Inlining:** The code can be compiled with function inlining enabled, which embeds smaller kernels within larger functions. While this results in a slightly larger binary, it reduces jumps in execution by minimising function call overhead. Additionally, inlining helps reduce the overhead of dereferencing memory pointers and improves data locality, leading to better utilisation of GPU memory bandwidth.
- **Optimised Reductions:** Modifications were made to handle data reductions within certain functions more efficiently for parallel execution. This optimization aims to enhance the reduction operations to better leverage parallel resources on the GPU, minimising unnecessary synchronisation and data movement where possible.

The primary challenge in demonstrating these improvements quantitatively is that the execution trace tools required for detailed performance analysis impose basic compilation requirements, which prevent more optimised

builds from being traced accurately. Nonetheless, these adjustments are expected to reduce kernel management overhead and improve the parallel efficiency of routine operations.

Next audits

In addition to a second code audit scheduled within ChEESE-2P WP2, FALL3D is leveraging POP3's advanced services, specifically Second Level Services (SLS). A proof-of-concept audit started in October 2024 to ensure compatibility of commonly used libraries, particularly MPI, across diverse MPI implementations and platforms. Once the code reaches full or near-full acceleration, an energy efficiency analysis will be conducted to identify further optimization opportunities. Finally, an advisory study will target specific code areas that could benefit from further improvements.

4.7.5 Other improvements

Enhanced Meteorological Data Handling

A chunk-based reading for large meteorological data was implemented in version 9.0 to enable very long simulations and allow for the processing of large meteorological files without requiring them to be fully loaded into memory. This approach significantly reduces the memory usage, preventing the huge memory usage and overflows found in older versions of FALL3D which severely limited the size of the problems and the resolution of the simulations. The so-called `MEMORY_CHUNK_SIZE` parameter introduced in the simulation configuration allows for control over the memory chunks allocated for meteorological data timesteps, offering to the users the ability to optimise the memory management based on hardware and simulation needs.

This method enhances code portability by supporting larger simulations across a broader range of computing systems, including those with constrained memory resources. While it may introduce a minor performance overhead, this chunk-based approach provides a flexible and scalable solution for managing large meteorological datasets.

4.7.6 Exascale ambition

In operational settings, FALL3D is typically used in ensemble-based runs where each ensemble member considers a single scenario (i.e. given values

of meteorological and volcanological input parameters) and the ensemble spans the ranges of uncertainties in model inputs. This approach is used both for probabilistic and deterministic forecasts and each ensemble member execution is independent from the remaining (no communication) model instances in a simple run (more advanced model configurations require communication between ensemble members). The ambition of the code is on capacity computing (embarrassing parallelism) and urgent computing, as reflected in the Simulation Cases addressed in ChEESE-2P (SC5.1 and SC5.2).

For a typical problem setup (grid span and resolution, number and characteristics of the particle bins), a few nodes are normally sufficient for each ensemble member to meet the operational time-to-solution constraints. The total ensemble node count grows quite linearly up to hundreds of ensemble members thanks to the efficient workflow embedded in the code: all the ensemble members can run simultaneously and almost independently (except from I/O operations) within a single job execution. As a result, the code ambition is focussed on reducing the time-to-solution of individual members rather than on strong scalability.

4.8 OpenPDAC

OpenPDAC is an open-source multiphase code for simulating volcanic gas-particle flows. Developed as a module of OpenFOAM, OpenPDAC builds upon the *multiphaseEuler* framework included with OpenFOAM. The code extends the original module by modifying the kinetic theory equations for granular flows to handle multiple dispersed solid phases and incorporating models for particle-particle drag interactions. OpenPDAC integrates a Lagrangian library (using one-way coupling with the gas-solid mixture) to simulate the behaviour of large solid particles. The code also includes functionality to initialize hydrostatic pressure profiles, a key feature for simulations on large computational domains.

4.8.1 Flagship code/Mini-app(s) initial status

The initial status of OpenPDAC at the beginning of the ChEESE-2P project can be seen by accessing the public repository: <https://github.com/demichie/OpenPDAC-10>. The version numbering is associated with the yearly releases of OpenFOAM by the OpenFOAM foundation (www.openfoam.org), with OpenFOAM 10 being released in July 2022. Since then two new versions of OpenFOAM and OpenPDAC have been released, with OpenPDAC 12 being the latest major release in July 2024

(<https://github.com/demichie/OpenPDAC-12>). We aim at releasing a new version of OpenPDAC within a few weeks after a major release of OpenFOAM is released.

In the first version, OpenPDAC followed the OpenFOAM approach, and it was developed as an “application solver”, containing a series of instructions that solved the governing equations within an iterative solution algorithm, and making function calls to various physical models, e.g. turbulence. The code was procedural, not object-oriented. Starting from version 11, and following the restructuring made by the OpenFOAM foundation, OpenPDAC is written as a “modular solver”, with the benefits of object orientation now extended to the solver component of OpenFOAM. Whereas application solvers present equations, algorithms and model interfaces in a procedural manner, a modular solver is built around a generalised solution algorithm residing in an external application (*foamRun*). Equations and interfaces are introduced at different levels of the class hierarchy and form the modular solver. With the last release, several improvements to the model have been added, both in the physics (a new energy equation, better description of the polydispersity of the particle), in the solution procedure (new checks on the residuals of the equation solution to reduce the number of iterations) and in the preprocessing (new utilities to generate the mesh). The repository contains the source code, a suite of test cases for multiphase gas-particles flows, and the utilities for the preprocessing. Each test case contains a script to perform in an automatic way the simulations.

Since version 11, the code is also available at the gitlab repository of the ChEESE-2P project (<https://gitlab.com/cheese5311126/codes>).

4.8.2 Porting

As the activity is on the CPU side only, Porting is not planned for OpenPDAC.

4.8.3 Performance Portability campaign

For the same reason, this activity is not planned for OpenPDAC.

4.8.4 Code Audit driven improvements

The code profiling analysis obtained during the first audit phase (although performed on a relatively low number of cores, less than 512, because the analysis traces had unmanageable file sizes for larger numbers of cores) highlighted well-known performance issues of OpenFOAM (Deliverable D2.1). The most relevant ones had also been reported in ChEESE for

ASHEE, a different OpenFOAM-based solver: memory bandwidth bottleneck (inducing a superlinear scaling) and low efficiency of inherent OpenFOAM communication patterns. In ChEESE, we demonstrated that a mixed-precision approach (Brogi et al., 2024, [10.1016/j.future.2023.10.006](https://doi.org/10.1016/j.future.2023.10.006)) reduces the impact of the memory bandwidth, but integrating such optimizations into the official Foundation release (www.openfoam.org) has not been possible so far. On the other hand, our choice is to avoid large “unsupported” interventions on the OpenFOAM numerical kernel and low-level libraries, and to remain coherent with the official releases, to guarantee the maximum portability of our solvers and the broadest usability by the geophysical community. For this reason, while we are continuously interact with the OpenFOAM community and developers (e.g., in the EU-funded exaFOAM project <https://exafoam.eu/>) to improve the parallel efficiency of the code, we focused our optimization efforts on ChEESE-2P on the specific solution algorithm of the OpenPDAC numerical solver (detailed in the following sections).

4.8.5 Other improvements

During the first Development access project on LUMI-C, the computational performances of OpenPDAC were successfully tested in terms of code scalability and computational speed up to 8192 cores. Two main issues affect the performance of OpenPDAC in this case: the mesh quality and the convergence constraints of the numerical solution algorithm, which are detailed in the following sections. Other algorithm improvements have been oriented to a more efficient physical description of the process (e.g., different treatment of the momentum exchange terms, and of the multiphase energy/enthalpy), and will be described elsewhere. Optimization aspects addressed in this report and described below have been exploited with a new EuroHPC Development access project on LUMI-C. It is worth recalling that, during the project, LUMI-C was unavailable for a long period, which reduced our testing and development capability.

Optimization of the numerical

In OpenFOAM (as in most CFD solvers), meshing cannot be considered just a pre-processing step, but it is part of the solution algorithm itself, because some of the numerical parameters for the spatial discretization and flux reconstruction schemes depends on metrics describing the mesh quality (face flatness, cell skewness, cell orthogonality, etc).

The original strategy for meshing at the beginning of the project was based on the use of the OpenFOAM utility *snappyHexMesh*, which generates 3-

dimensional meshes containing hexahedra (hex) and split-hexahedra (split-hex) from triangulated surface geometries. The mesh approximately conforms to the surface (the topography in our case) by iteratively splitting the cells of a uniform base mesh and morphing the resulting split-hex mesh to the surface.

- The simulations performed during the first audit stage have shown that halving the cell size of the base mesh and then using *snappyHexMesh* causes a reduction of the time step by a factor of 10, instead of 2 (expected from the CFL condition).

This is mostly associated with the “split” step of *snappyHexMesh*, which can create arbitrarily small cells resulting in a small-time step (according to the CFL condition). In addition, a large number of non-hexahedral cells are produced, mostly located close to the topography, which is the region where an accurate solution is required for a proper modelling of pyroclastic flows. These cells, mostly prismatic and tetrahedral, cannot guarantee a proper discretization of the advective fluxes. For these reasons, we designed *from scratch* a new meshing tool, which, on one hand, allows us to generate fully hexahedral meshes, and on the other gives a better control on the minimum size of the cells. The new meshing strategy is based on the creation of a fully hexahedral mesh for a flat topography with a crater, obtained with the union of several blocks to have: 1) the desired uniform horizontal size of the cells on the topography; 2) the desired grid refinement in the crater area; 3) a coarsening of the mesh in the upper part of the domain. Then, the mesh is deformed with a newly created utility, *topoGrid*, to have the lower face conformal to the topography, while keeping the hexahedral topology of the cells and avoiding distortions that can deteriorate the quality of the cells. One common problem with the generation of hexahedral cells conformal to a prescribed surface is that in general it is not possible to *tassellate* a surface with planar quad cells. This requires, within the utility *topoGrid*, a preliminary step to maximise the flatness of the topography quads and avoid the generation of hexahedral cells like those shown in Fig. 4-8.1.

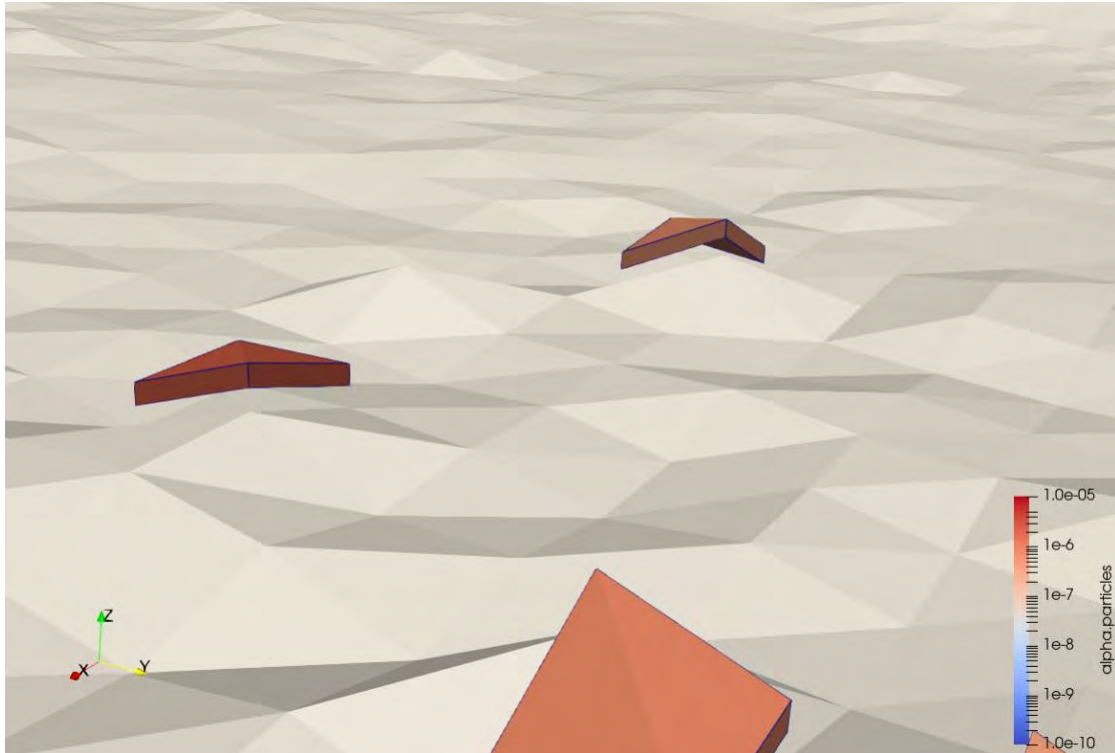


Fig. 4-8.1: Cells produced when the topography mesh points generate non-planar quads. The resulting hexahedral cells may have the centroid outside the cell, resulting in a problematic decomposition in tetrahedra.

For the internal mesh points, the deformation algorithm is based on an explicit weighted inverse distance interpolation, utilising a two-exponent form of the weighting function to preserve near-boundary deformations while providing a smooth transition in the interpolation to blend deformations in the bulk of the volume mesh. We also use the area of the surface facet in the weighting function so that mesh refinement of a region (like in the crater area) does not increase its influence in the interpolation method.

Here we report the output of the OpenFOAM utility checkMesh, which provides an analysis of the quality of the mesh generated by *topoGrid*.

Overall number of cells of each type:

```
hexahedra:      41256376
prisms:         0
wedges:         0
pyramids:       0
tet wedges:     0
tetrahedra:     0
polyhedra:      0
```

Max aspect ratio = 14.3820624236 OK.

Mesh non-orthogonality Max: 65.2645680258 average: 23.7032941107

Non-orthogonality check OK.
Max skewness = 1.47181542737 OK.
Face tets OK.
Face flatness (1 = flat, 0 = butterfly) : min = 0.992060370889 average = 0.999994112026
All face flatness OK.
Concave cell check OK.
Face volume ratio : minimum: 0.63802266543 average: 0.992399469279
Face volume ratio check OK.

It is apparent that only hexahedra are generated, and that the most important quality parameters are within the threshold for a good mesh. The explicit nature of the meshing algorithm is well suited for a parallel implementation. This has been obtained by writing the mesh deformation tool as a C++ utility, leveraging the classes and libraries provided by OpenFOAM. This approach allowed us to take advantage of OpenFOAM's parallelization capabilities, ensuring efficient performance and scalability. In any case, the computational cost of the explicit weighted inverse distance interpolation remains high, increasing with $N_s \times N_i$, where N_s is the number of topography mesh points and N_i is the number of internal mesh points. Thus, when the horizontal cell size is halved, the computational cost increases by a factor 2^5 and, in addition to the parallelization of the interpolation utility, further optimization could be beneficial. In any case, when the preprocessing is performed in parallel, its computational cost is already not significant with respect to that of the simulation with OpenPDAC. The new preprocessing utility has been tested also on ARM parallel processors.

The new mesh guarantees a proper scaling of the number of cells with changes in the minimum size, and thus also a correct scaling of the theoretical time step required for the stability of the numerical schemes. As a result of the increase of the time-step, the net gain in the time-to-solution is about a factor 5.

Optimization of the numerical solution algorithm

OpenFOAM solvers are based on a segregated approach and utilise the PIMPLE algorithm to couple the governing equations for mass, momentum, and energy in transient simulations. This algorithm is structured using nested loops: an outer PIMPLE loop, an inner PISO loop, iterative corrections for temperature and the mass fractions of the components of the different phases, and additional iterative corrections to handle non-orthogonality. In the original OpenFOAM implementation, the number of iterations for each loop is constant during a simulation and fixed by the

user through an input file. In OpenPDAC, we have introduced the capability to exit each loop when the desired residual in the solution of the equations is achieved. The improvement of the convergence can significantly reduce the number of equation solutions required for each time step, leading to a substantial optimization (about a factor 2) in computational performance (Fig. 4-8.2).

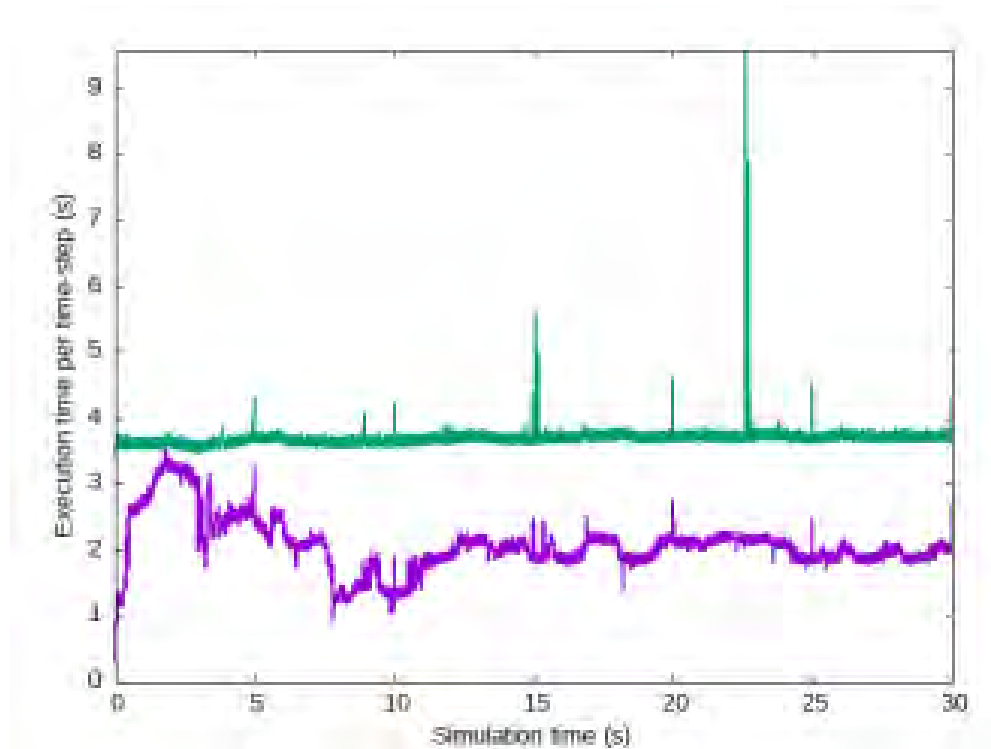


Fig. 4-8.2: Overall effect of the optimisation on the numerical solution algorithm (residual checks to optimize the number of iterations). Except for the first 5 seconds, the execution time and therefore the time-to-solution is halved (purple line)

4.8.6 Exascale ambition

Being based on OpenFOAM, OpenPDAC inherits the parallel MPI capabilities of the original library, having efficient (i.e., greater than ~ 0.75) parallel scalability up to (10^4) CPU cores, with optimal performances achieved in the range $(5-10 \times 10^3)$ finite-volumes per core. This baseline allows us to efficiently address parallel supercomputers' volcanic flow simulations with up to 10^8 mesh elements and several multiphase constituents.

The exascale challenge addressed in ChESEE-2P is thus twofold: 1) reducing the time-to-solution and setting up Urgent Computing workflows for Simulation Cases relevant for hazard assessment during volcanic crises; 2) performing Capacity numerical simulation ensembles (with 10^2-10^3 instances) to quantify uncertainties and perform (short- and

long-term) Probabilistic Volcanic Hazard Assessment (PVHA) for phreatic volcanic eruptions.

To address the above exascale challenges, the optimization of the parallel convergence of the solution algorithm in the compressible, turbulent, multiphase flow regime characterizing volcanic explosions has been addressed, and the meshing strategy and the domain decomposition approach have been improved, as described above. Ongoing work suggests that we will achieve a significant reduction in the time to solution, mainly thanks to much larger timesteps (Fig. 4-8.3). An analysis of the other components of the parallel workflow for capacity computing has just been addressed and optimization tasks are still to be planned in WP4.

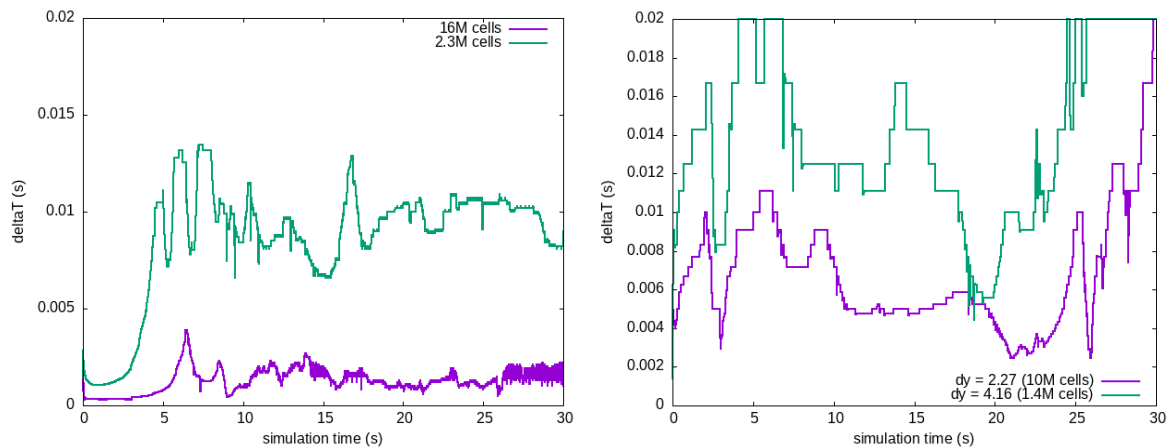


Fig. 4-8.3: On the left, we report the time step evolution for two simulations with different mesh resolutions for phreatic eruption including the volcano topography. With the new mesh generator strategy (right) halving the cell size causes the time step to be reduced by a factor of 2, instead of the factor 10 observed with the previous code version. The net gain is around 5 on time-to-solution.

4.9 LaMEM

LaMEM is a code to simulate geological processes. So far it was an MPI-parallelized CPU-based code written on top of PETSc (<https://petsc.org>). One primary reason for this is the evolving state of PETSc's accelerator support, which is in active development.

4.9.1 Flagship code/Mini-app(s) initial status

Each PETSc version update introduces changes to workflows, available runtime options, and internal kernels, making it challenging to maintain a stable and consistent setup for accelerator use. Additionally, the LaMEM development team had limited access to accelerated HPC systems, which has hindered testing and development on GPUs. Initial tests using GPU were conducted on a local machine with NVIDIA GPU on M13 and demonstrated performance improvement in certain kernels. Since 2023, the team has access to MOGON-NHR (see for example <https://nhrsw.de/systems/>), a local HPC cluster with NVIDIA accelerators, followed by EuroHPC resources on LUMI-G, equipped with AMD GPU hardware, in M14 of the project.

Another significant challenge has been installing the latest versions of PETSc (<https://arxiv.org/pdf/2406.08646>) on various HPC clusters. Different platforms require specific configurations and combinations of packages tied to the unique hardware and software environment, often demanding system administrator-level effort. Waiting for support from HPC teams to install and configure the necessary libraries and packages (e.g., on LUMI) resulted sometimes in substantial delays, slowing development and testing.

Through participation in the ChEESE project and engagement with work packages focused on porting activities, the LaMEM development team aims to achieve performance portability for the code. This includes streamlining workflows for installation and enabling multi-node GPU runs across different HPC platforms. The primary goal is to explore the feasibility and efficiency of using both NVIDIA and AMD GPUs for LaMEM simulations, comparing their performance against traditional CPU-based runs to assess potential gains in computational efficiency.

4.9.2 Porting

One major advantage of using PETSc in LaMEM is that the core structure of the code remains consistent across CPU and GPU platforms. Functions for

data manipulation, such as solver and matrix operations, are designed to work seamlessly between CPU and GPU memory, preserving the integrity of the codebase while allowing for optimization on both current and future hardware architectures.

PETSc's design separates the programming models of external packages from its internal model, sharing data but avoiding unnecessary memory copies and thus preserving efficiency. This separation enables PETSc users to choose among different programming models, such as CUDA, HIP, or Kokkos, which in turn allows for flexibility in selecting the computational platform (e.g., HIP for AMD GPUs) without changing the underlying code. Users can specify platform-specific backends for matrix and vector operations at runtime by setting options such as `--vec_type` for vectors and `--mat_type` for matrices from the command-line, configuring GPU-specific objects at runtime.

In recent versions, PETSc introduced a robust blueprint for porting applications to GPUs (<https://publications.anl.gov/anlpubs/2020/04/159190.pdf>), enhancing compatibility and extending support across various architectures. The PETSc/CUDA backend has been optimized with features that improve performance, and the PETSc/HIP backend has also seen improvements to support AMD GPUs more effectively. The PETSc/Kokkos backend, which is vendor-neutral, provides flexibility by supporting a range of GPUs and CPUs, including devices from NVIDIA, AMD, and Intel.

Additionally, PETSc's star forest (*PetscSF*) abstraction for communication patterns offers an efficient means for inter-device communication, avoiding explicit synchronization across GPUs. This is particularly beneficial given the lack of a standardized GPU MPI implementation in current software environments. Due to the high memory bandwidth of GPUs, message passing for data packets larger than 1MB often achieves lower latency than comparable CPU-based operations, enhancing overall computational efficiency in large-scale, multi-node simulations.

4.9.3 Performance Portability campaign

Performance campaign on local hardware

The status of performance portability in LaMEM is shown through a scaling study of the 3D "Falling Spheres" model, which simulates falling spheres using the Navier-Stokes equations. All models were executed to the

convergence of a single non-linear iteration, using symmetric resolutions in all three dimensions (e.g., 64x64x64 or 128x128x128) and a fixed multigrid level of 3. The largest model, with a resolution of 256x256x256, requires approximately 315 GB of RAM, which is the maximum feasible on a typical HPC node. This memory limitation restricts our ability to fully explore GPU performance on setups with 2-8 GPUs, as GPUs achieve better efficiency with longer vectors.

Initial tests were conducted on a local machine equipped with 2 x AMD EPYC 7543 processors (64 cores in total) and 512 GB of RAM, as well as two NVIDIA A100 GPUs with 80 GB of VRAM each, where we utilized cusparse vectors and matrices. This work is focused on offloading the most computationally intensive and time-consuming operations, specifically matrix and vector multiplications, represented by the PETSc functions *MatMult* and *VecAXPY*.

In Fig. 4-9.1 (left), we compare the execution time of *MatMult* for runs with 2 and 64 MPI ranks versus a configuration with 2 MPI ranks using 2 GPUs. Results show that even when compared to all 64 CPU cores, the 2-GPU configuration achieves a notable speedup. Fig. 4-9.1 (right) illustrates the time spent on the *VecAXPY* function, with higher-resolution models demonstrating increased speed-up on GPUs, suggesting that the GPUs have not yet reached their maximum performance.

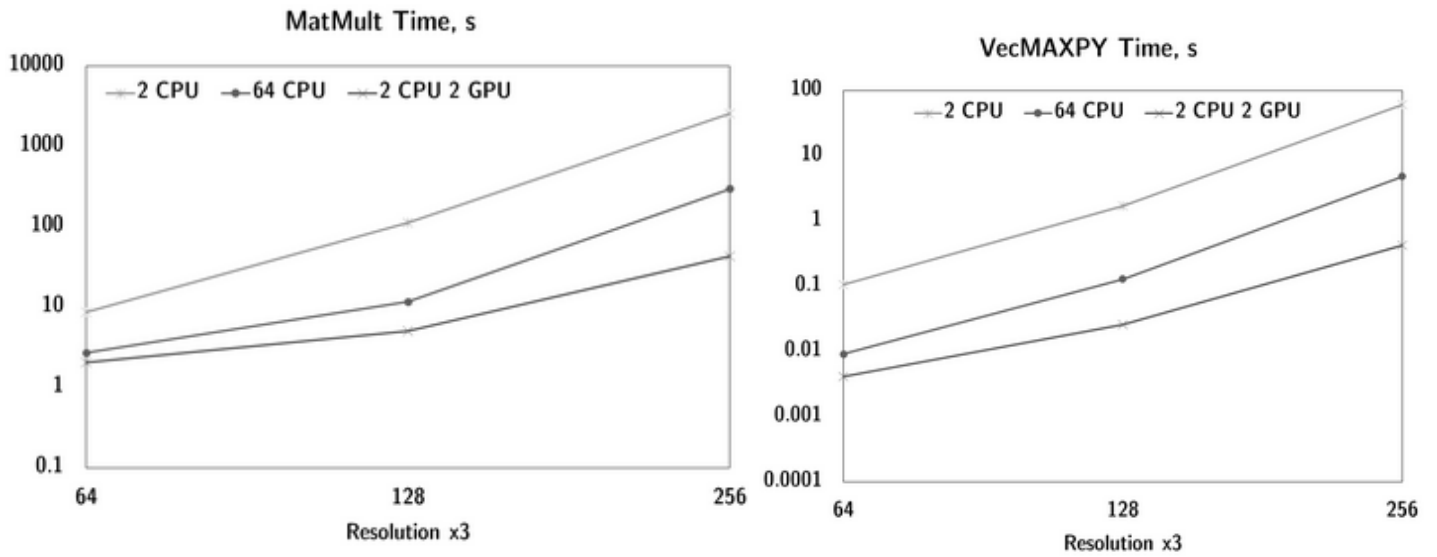


Fig. 4-9.1: Left: MatMul execution. Right: time spent on VecAXPY function

In Fig. 4-9.2, we assess performance by comparing the number of operations per second for each function. CPU-only runs with 2 and 64 cores show a plateau, indicating they have reached their maximum bandwidth or peak performance. In contrast, GPU-based runs continue to scale, even up to the highest resolution of 256x256x256, suggesting further performance potential. However, we encountered a limitation due to available memory, restricting further scaling. This highlights a potential area for future exploration, such as the use of virtualized GPUs to extend memory capacity and improve utilization further.

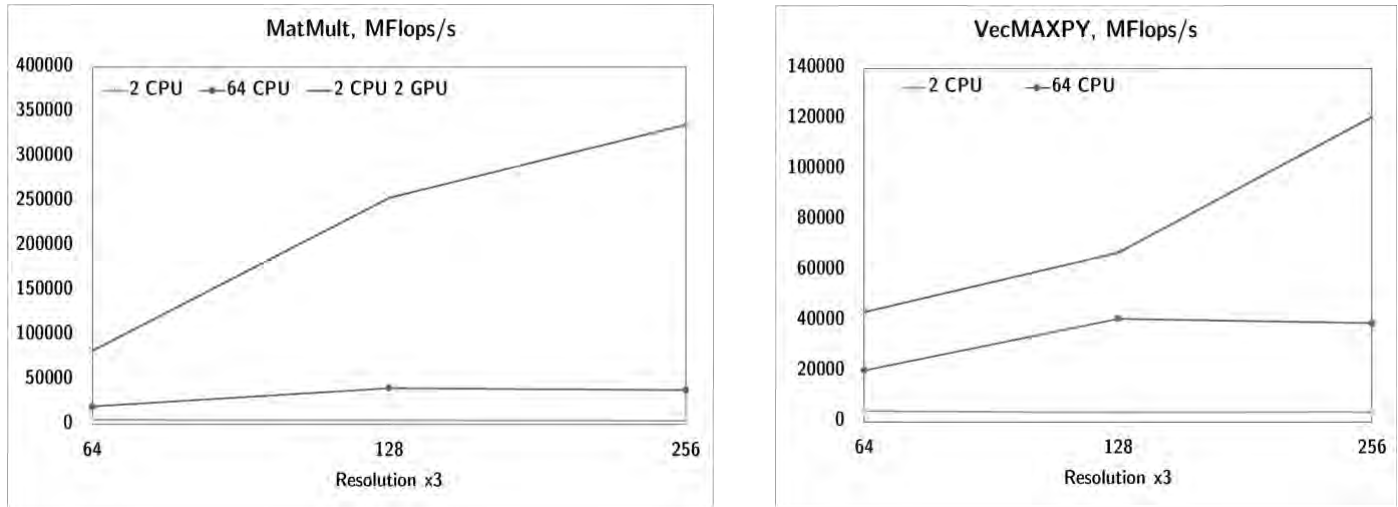


Fig. 4-9.2: Performance in MFLOP/s Left: MatMul, Right: VecAXPY.

Performance campaign on LUMI-G

The EuroHPC cluster LUMI-G has a distinct node architecture, with each node featuring a single AMD EPYC "Trento" processor, 8 x 64 GB RAM totalling 512 GB, and 4 AMD MI250x GPUs. Each MI250x GPU includes two Graphics Compute Dies (GCDs). Efficient distribution of MPI communicators is crucial on this architecture, as only every 8 cores share an L3 cache—a factor that plays a significant role in achieving optimal GPU bandwidth through the Infinity Fabric. This setup is further illustrated in Fig. 4-9.3 from the LUMI documentation (<https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/lumi-g-job/>).

To optimize L3 cache distribution across GPUs, we specify binding using the following core map: 49, 57, 17, 25, 1, 9, 33, 41, with every first core in each L3 region reserved for the operating system. PETSc has been configured with HIP support on LUMI, allowing for GPU offloading to AMD devices. Accordingly, matrices and vectors are set to the *hip* and *mpiaijhipspare* types for proper GPU handling.

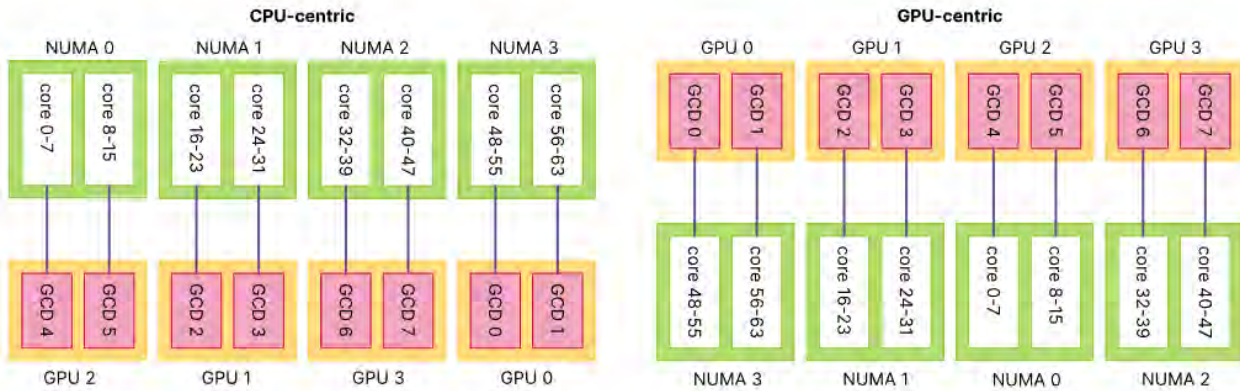


Fig. 4-9.3: CPU-GPU links from a CPU centric or GPU centric point of view

Fig. 4-9.4 compares performance between runs using 8 CPUs alone and those using 8 CPUs with 8 GPUs. Due to overhead, GPU runs only demonstrate significant speedup at higher resolutions where longer vectors are processed. Beyond a resolution of 128x128x128, we observe faster runtimes with GPU assistance, underscoring the advantage of GPUs in handling larger data sets.

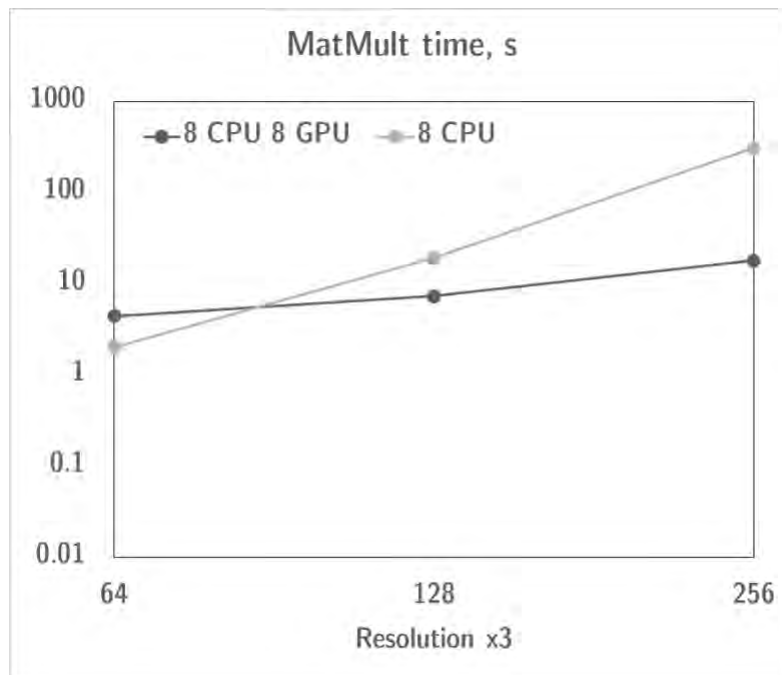


Fig. 4-9.4: MatMul time, CPU vs. GPU

Like results on NVIDIA devices, GPUs here show performance benefits primarily when processing longer vectors. Fig. 4-9.5 and further illustrate this, displaying operations per second. While CPU-only runs reach a performance plateau, GPU performance continues to improve with

increased vector and matrix sizes, highlighting the value of GPUs in scaling up LaMEM simulations on LUMI's architecture.

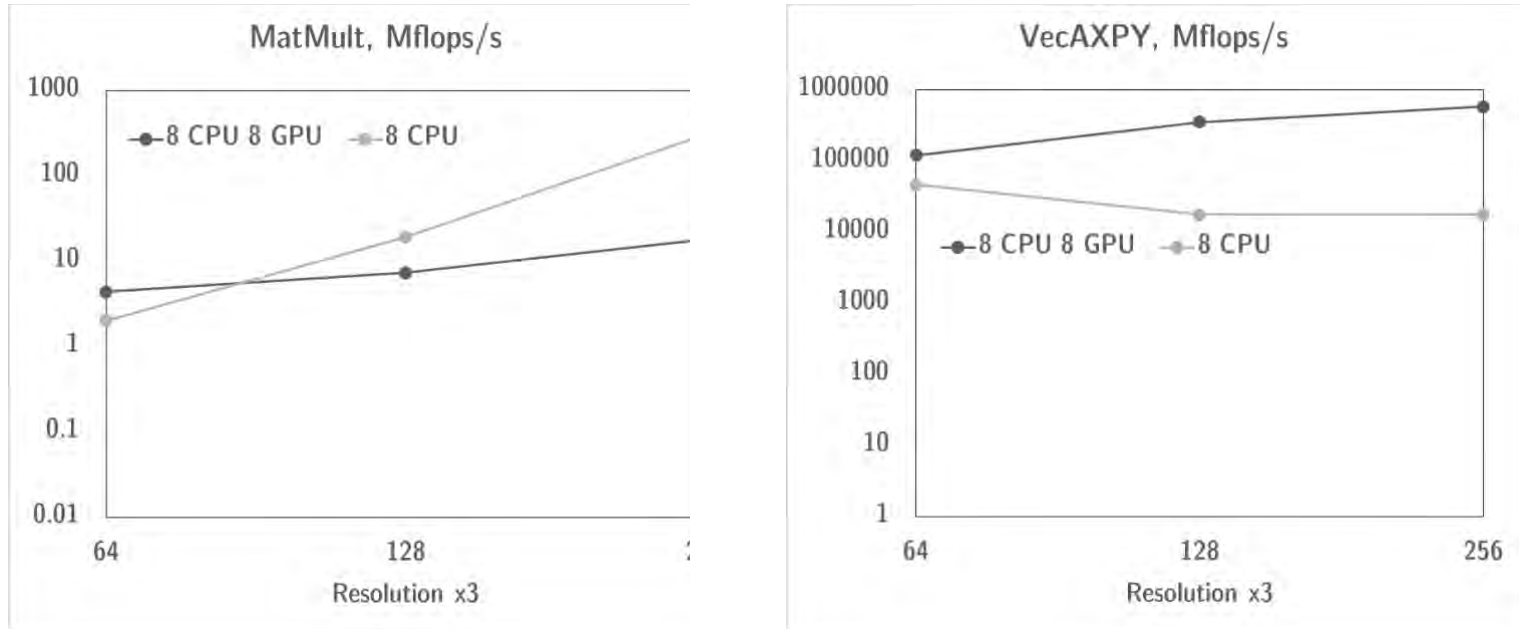


Fig. 4-9.5: MatMul (left) and VecAXPY (right) performance in MFLOP/s. CPU vs. GPU.

This report presented the outcomes of porting LaMEM to GPU-accelerated platforms, utilizing both NVIDIA and AMD devices. We detailed PETSc's design for GPU portability and explained how LaMEM leverages these capabilities to achieve performance gains across different architectures. Our findings show that GPU integration becomes particularly advantageous at higher resolutions and larger data sizes, underscoring the benefit of GPUs over CPU-only runs in these scenarios. These results validate the effectiveness of our porting efforts and demonstrate LaMEM's adaptability and performance potential on diverse HPC architectures.

In the most recent edition of TOP500 (www.top500.org), ranking the 500 most powerful supercomputer in the world and published in November 2024, LUMI and Leonardo are the two EuroHPC clusters in the first ten positions, respectively 8th and 9th.

Focusing on the accelerated partition, one LUMI-G computational node features 4 AMD Instinct MI250X GPUs (2nd Gen AMD CDNA architecture), each with 220 Compute Units for a theoretical peak performance in double

precision (FP64) of 47.9 TFLOP/s, together with 128 GB of HBM2e memory offering over 3.2 TB/s of memory bandwidth.

However, the MI250X GPU is a multi-chip module with two GPU dies named by AMD Graphics Compute Die (GCD). Each GCD has access to a 64 GB slice of HBM2e memory and 110 compute units for a FP64 theoretical peak of 23.95 TFLOP/s.

From a software perspective as well as from the Slurm perspective, one GCD is considered as one GPU for a total of 8 GPUs per node. In the following, for LUMI-G we adopt this perspective and regard one GCD as one GPU.

Running the OpenCL-Benchmark (<https://github.com/ProjectPhysX/OpenCL-Benchmark>) on one GPU (i.e., one GCD), we report an empirical peak performance for FP64 of 16.94 TFLOP/s.

For Leonardo Booster, one computational node features a single Intel Xeon 8358 32 cores (2,6 GHz), 512 (8 x 64) GB RAM DDR4 3200 MHz, and four NVIDIA custom Ampere A100 GPU (each with 64GB HBM2 and featuring 124 streaming multiprocessors, 1640 GB/s of memory bandwidth and peak FP64 performance of 11.2 TFLOP/s).

Running the OpenCL-Benchmark on one Leonardo Booster GPU, we report an empirical peak performance for FP64 of 10.9 TFLOP/s. Moreover, with the continuous development of accelerator technology, it is worthy to test LaMEM also on the very recent NVIDIA GH200, thus investigating the possible speed-up offered by this new generation of accelerators.

With this variety of available hardware, in this section we present LaMEM performances on the EuroHPC clusters LUMI-G and Leonardo Booster, as well as the NVIDIA multi-node Grace-Hopper Evaluation System (Thea) based on NVIDIA Quanta S74G-2U GH200. Specifically, each Thea node consists of a Grace CPU 72 cores (3.4GHz) with 480 GB LPDDR5X memory and a single Hopper GPU with 96 GB HBM3 (FP64 theoretical peak performance of 34 TFLOP/s).

Lastly, running the OpenCL-Benchmark on one Thea Hopper GPU, we report an empirical peak performance for FP64 of 31.26 TFLOP/s.

Since the focus of the Performance Portability campaign is on accelerator performances, we need to reduce the effect of the network as well as the differences in the CPU architectures.

For this reason, we always consider tests running within one single computational node avoiding inter-node communications.

Since Thea has only one GPU per node, to have a fair comparison, we always considered tests running on a single GPU also on LUMI-G and Leonardo Booster.

Moreover, each cluster has a very different CPU configuration in terms of architecture, number of cores and frequency. Thus, following the same principle, we choose to run the tests using only one CPU core, selecting one MPI Rank (one task) and one shared memory Thread (one cpu-per-task).

In order to fully explore several offloading possibilities offered by PETSc (version 3.22.0), we consider CUDA (*vec_type cuda* and *mat_type aijcusparse*) and Kokkos (*vec_type kokkos* and *mat_type aijkokkos*) backends on NVIDIA based clusters (Leonardo Booster and Thea Grace-Hopper), whereas we consider HIP (*vec_type hip* and *mat_type aijhipsparse*) and Kokkos backends on LUMI-G, focusing on a scaling study of the pure "Diffusion" model, with no mechanics involved.

As in the previous section, we consider a symmetric resolution in all three dimensions: a small size problem with resolution 64x64x64, an intermediate problem 128x128x128, and the largest problem 256x256x256. Moreover, we set the multigrid preconditioning level to 5. Metrics have been collected using the internal PETSc profiler using the runtime logging options *-log_view -log_view_gpu_time*.

Specifically, we report (Fig. 4-9.6) the GPU MFLOP/s (sum of MFLOP/son GPU / max GPU time) for the *MatMult* and *VecMAXPY* offloaded kernels as measured by PETSc (mind that these are the average results of two runs without any attempt to optimize performance on a particular platform, with no statistics applied).

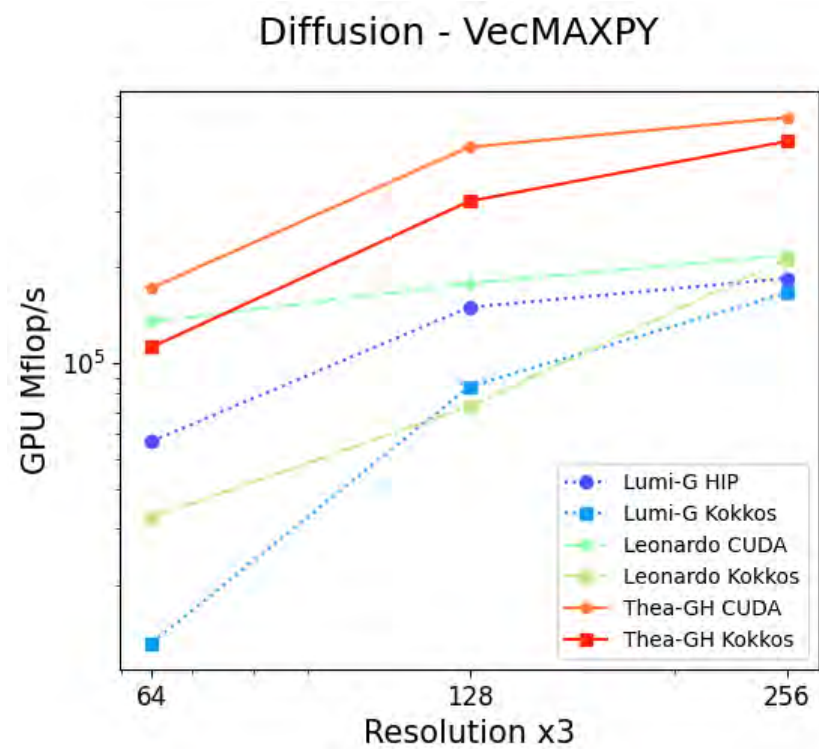
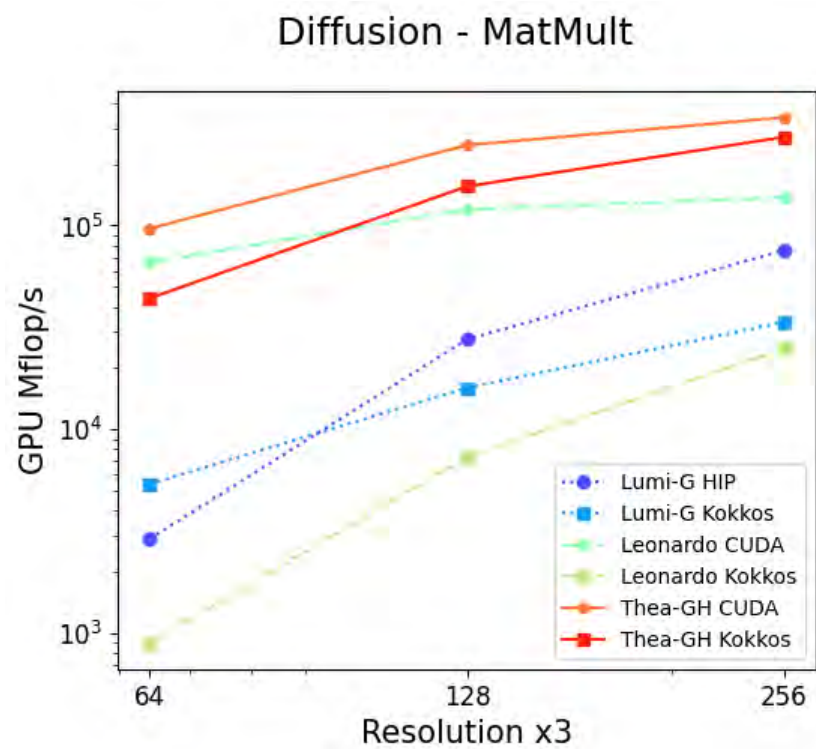


Fig. 4-9.6: MatMul and VecAXPT performance using different HW

We report the GPU MFLOP/s (sum of MFLOP on GPU / max GPU time) for the MatMult and VECMAXPY offloaded kernels as measured by PETSc (mind that these are the average results of two runs without any attempt to optimize performance on a particular platform, with no statistics applied). Moreover, we report the architectural efficiency as the achieved performance in terms of FLOP/s rates as a fraction of peak hardware performance (both in terms of the theoretical peak performance as well as the empirical peak performance, see Tab. 4-9.1, 4-9.2 and 4-9.3).

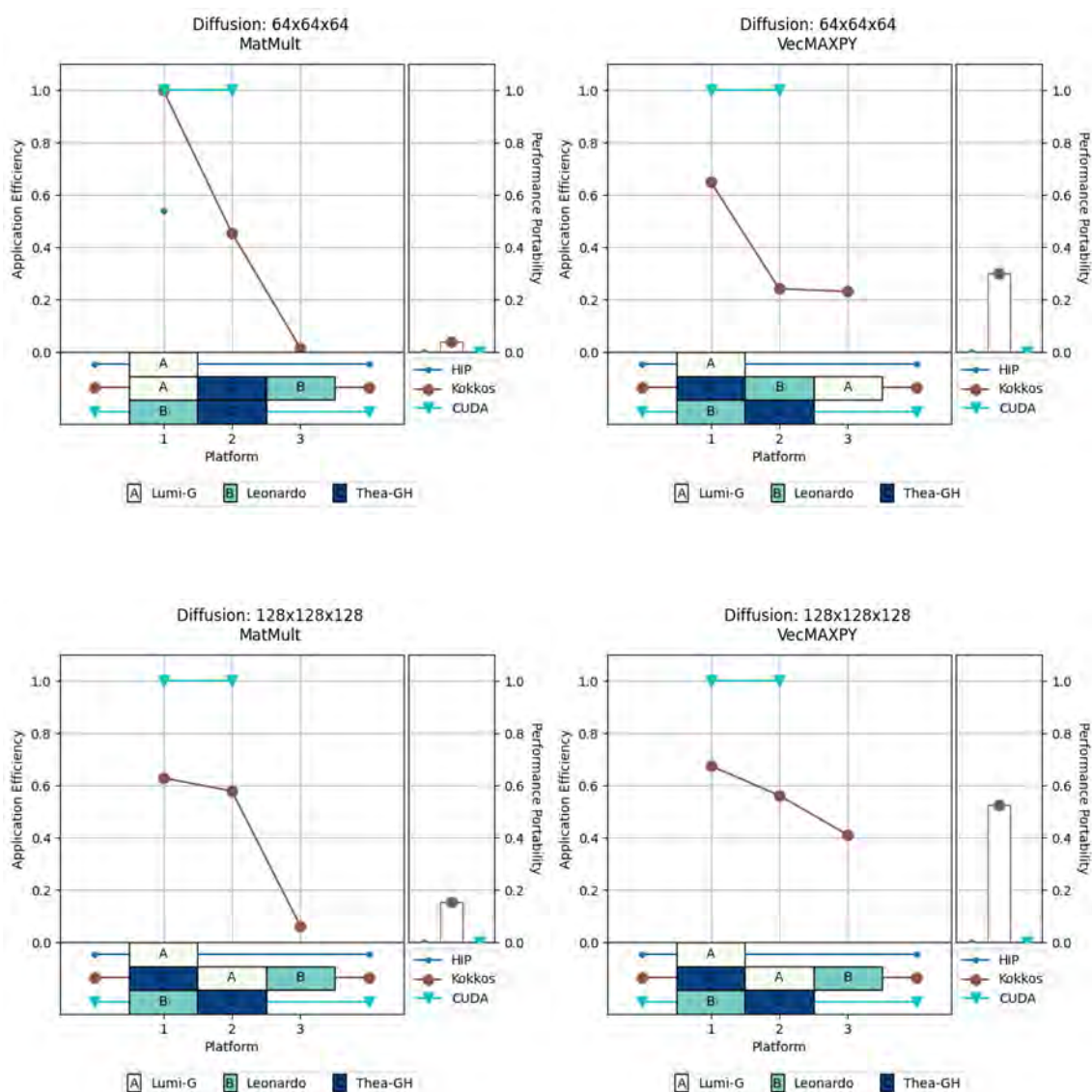
Resolution 64x64x64	MatMult GPU MFLOP/s	MatMult Theor. Arch. Eff.	MatMult Empir. Arch. Eff.	VecMAXPY GPU MFLOP/s	VecMAXPY Theor. Arch. Eff.	VecMAXPY Empir. Arch. Eff.
Leonardo Kokkos	897	0.008%	0.008%	32542	0.290%	0.298%
Leonardo CUDA	66108	0.590%	0.606%	135086	1.206%	1.239%
LUMI-G Kokkos	5392	0.022%	0.031%	13080	0.054%	0.077%
LUMI-G HIP	2905	0.012%	0.017%	56661	0.236%	0.334%
Thea-GH Kokkos	43932	0.129%	0.140%	112185	0.329%	0.358%
Thea-GH CUDA	96541	0.283%	0.308%	172586	0.507%	0.552%

Resolution 128x128x128	MatMult GPU MFLOP/s	MatMult Theor. Arch. Eff.	MatMult Empir. Arch. Eff.	VecMAXPY GPU MFLOP/s	VecMAXPY Theor. Arch. Eff.	VecMAXPY Empir. Arch. Eff.
Leonardo Kokkos	7315	0.065%	0.067%	72968	0.651%	0.669%
Leonardo CUDA	119985	1.071%	1.100%	178031	1.589%	1.633%
LUMI-G Kokkos	15989	0.066%	0.094%	84263	0.351%	0.497%
LUMI-G HIP	27648	0.115%	0.163%	150075	0.626%	0.885%
Thea-GH Kokkos	155431	0.457%	0.497%	323668	0.951%	1.035%
Thea-GH CUDA	247594	0.728%	0.792%	479829	1.411%	1.534%

Resolution 256x256x256	MatMult GPU MFLOP/s	MatMult Theor. Arch. Eff.	MatMult Empir. Arch. Eff.	VecMAXPY GPU MFLOP/s	VecMAXPY Theor. Arch. Eff.	VecMAXPY Empir. Arch. Eff.
Leonardo Kokkos	24973	0.222%	0.229%	213164	1.903%	1.955%
Leonardo CUDA	137202	1.225%	1.258%	218624	1.952%	2.005%
LUMI-G Kokkos	33595	0.140%	0.198%	166701	0.696%	0.984%
LUMI-G HIP	75535	0.315%	0.445%	184946	0.772%	1.091%
Thea-GH Kokkos	271630	0.798%	0.868%	500554	1.472%	1.601%
Thea-GH CUDA	338782	0.996%	1.083%	595028	1.750%	1.903%

Tab. 4-9.1/2/3: Architectural Efficiency for the three different test cases.

Finally, since at least two implementations are available for each architecture (HIP and Kokkos on AMD, CUDA and Kokkos on NVIDIA), we report also the Application efficiency, and the Performance Portability obtained using the Performance, Portability and Productivity Analysis Library.



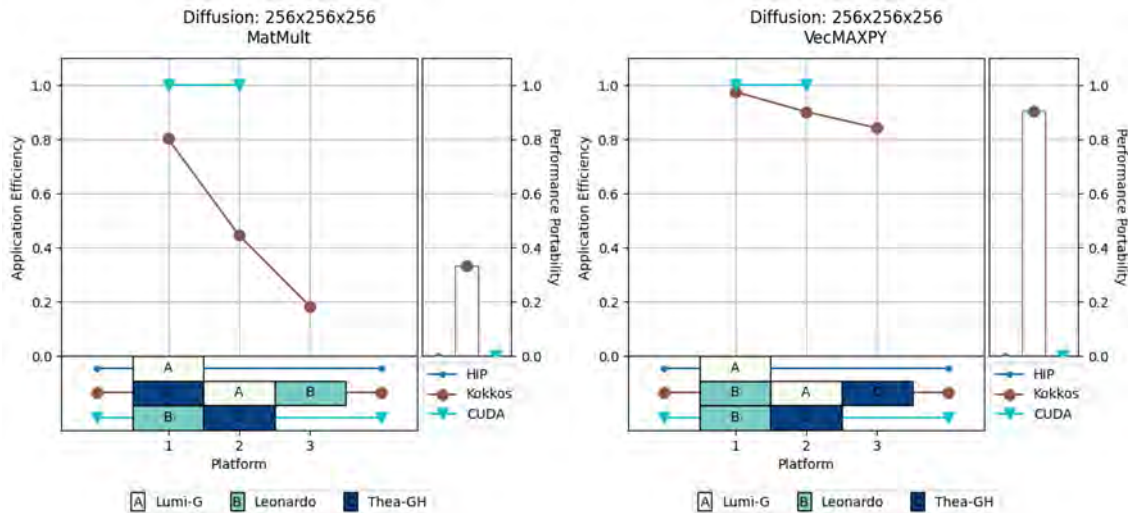


Fig. 4-9.7: Performance Portability Campaign for the different resolutions.

4.9.4 Code Audit driven improvements

The code audit demonstrated problems with scaling the coarse grid solvers used in LaMEM to large number of nodes on the CPU-only machines. We have made improvements in this direction, which will be reported elsewhere as they do not concern the portability of GPUs (which is the only focus of the report in D2.2). Since LaMEM was not part of the first phase of CHEESE, the code was not yet optimized or even tested for GPUs at the start of CHEESE-2P project and we therefore have quite some catch-up works to do in this respect. We can therefore only devote part of our time to the GPU porting, while much of the effort goes into making new matrix-free solvers available in LaMEM. In principle, we believe that this will improve the weak scalability of the overall code.

4.9.5 Other improvements

Nothing to report for LaMEM.

4.9.6 Exascale ambition

The results shown here demonstrate that LaMEM does in principle work on GPU accelerators, even without major code modifications. We expect that the matrix-free solvers, which we are currently being added to LaMEM, may help the overall performance along with the parallel scalability of the code, and we hope that this translates in benefits for the GPU-part as well. If that is the case, we will potentially be in a situation where we can use exascale machines along with their accelerators towards the end of this project. It is however not guaranteed that this works out of the box, as the matrix-free

innermost calculations make less use of the internal PETSc *Vec* and *Mat* objects, which have already been ported to the various GPU architectures. Future work will focus also on employing multiple CPUs per GPU (using virtualization) and analysing the overhead introduced by GPU use, especially in simulation initialization. Since GPUs are separate devices with their own communication mechanisms, some initialization latency is expected, and a deeper understanding of this overhead will help refine our approach.

Additionally, a preliminary analysis indicates that the matrix and vector assembly stage warrant further examination. This phase may be ported to GPUs, potentially using coordinate-based (COO) matrix assembly to optimize the insertion of values into preallocated global matrices. By addressing these areas, we aim to enhance LaMEM's performance on GPU architectures and streamline its operation across diverse hardware platforms.

If we instead must rewrite our innermost compute kernels to use Kokkos or other abstraction layers, it will likely be too time-consuming to do with the personnel months allocated to the project. As explained in the original ChEESE-2P proposal, the Scientific Challenges will be performed on the CPU-only parts of existing machines.

4.10 pTatin3D

pTatin3D is an open-source software package designed to study long time-scale processes in geodynamics. Originally developed to enable high-resolution, three-dimensional modeling of lithospheric deformation, it provides the geodynamics community with a powerful and accessible toolkit. The software features fast, parallel, and scalable matrix-free implementations of the Stokes operators, which are efficiently solved using a hybrid geometric-algebraic multigrid preconditioner. These capabilities make pTatin3D particularly well-suited for tackling complex geodynamic simulations at high resolutions.

4.10.1 Flagship code/Mini app(s) initial status

pTatin3D is a new flagship code that did not participate in ChEESE original project. The code was never audited officially before but was used on different HPC machines in Europe with good scaling on CPU architecture. The GPU support was partly there with some testing on Piz Daint but never used in production.

The pTatin3D mini app is new and was developed within the frame of ChEESE-2P. It consists of a simplified simulation case “model” that uses the flagship code to solve one time step of a linear problem. The heterogeneities of the coefficients in the simulation case are coarse enough to be captured with different resolutions, and large enough to test the robustness of the linear solver. This allows testing the different kernels on different platforms.

4.10.2 Porting

Before ChEESE-2P, pTatin3D already had kernels available for CPU x86_64 AVX2 and GPU NVIDIA written with CUDA. To improve the portability of the code on GPU, we added a new kernel (SpMV MF) written with HIP to use AMD GPUs. Currently, the kernel is up-and-running, and its development is still on-going.

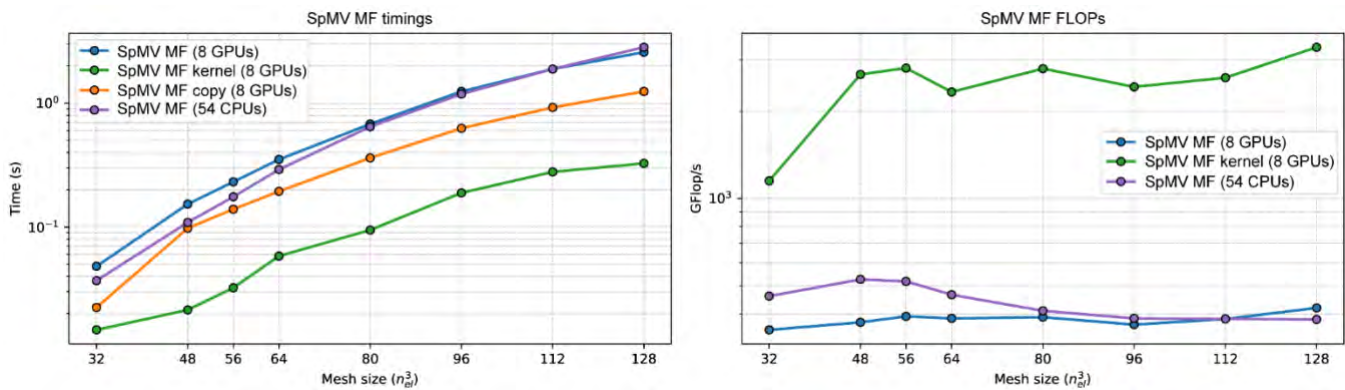


Fig. 4-10-1: spMV MF timings (left) and GFLOP/s (right) obtained for a given benchmark

4.10.3 Performance Portability campaign

Fig. 4-10.1 shows the performance of the HIP kernel on a single node (8 GPUs) on the LUMI-G supercomputer versus the performance of the CPU kernel on a single node (54 CPUs). The yellow curve shows the time spent to copy data from the CPU memory to the GPU memory which is currently the bottleneck of the HIP kernel.

There is no real performance portability activity to report, as pTatin3D needs a specific PETSc version to be ported to other architectures (such as the recent NVIDIA GPUs available on Leonardo or MareNostrum 5 requiring CUDA 11.8 or newer). Team is actively working with CINECA to solve this issue and guarantee at least the porting to EuroHPC NVIDIA platforms.

4.10.4 Code Audit driven improvements

During the first code audit, a bottleneck was identified on the parallelisation. The time spent by all the MPI ranks waiting for a single core operation to complete was too high. We greatly improved this issue with a better repartition of the coarse grid problem. We did not audit the code again, but internal tests show a better scaling.

As for the HIP kernel bottleneck reported in the section 4.10.2, this was monitored after the code audit, and we are currently working on the improvement of this kernel (section 4.10.5).

4.10.5 Other improvements

There is no activity to be reported here. Anyway, we plan on developing a PETSc Vec implementation which simultaneously holds part of the Vec on CPU and GPU. The Interior of subdomain will be stored on GPU and boundary and halo will be stored on CPU because it is the only part that needs to be copied back and forth between CPU and GPU. This should improve the time spent copying data and beneficial to code, mini app and kernel performances.

4.10.6 Exascale ambition

The activities of pTatin3D in the project are related to work in the SC 8.1 of ChEESE-2P, within which we plan to solve problems on a mesh with 135 million of nodes and 4 degrees of freedom (DOFs) derived from the incompressible Stokes problem, resulting in a system with over 500 million of DOFs to be solved at each time step. Since we deal with time-dependent problems, this operation needs to be repeated thousands of times, and the work in WP2 and within the project will allow us to leverage a massive number of resources from the LUMI supercomputer to achieve the goal.

4.11 ELMER/ICE

ELMER/ICE is a ChEESE-2P newly introduced code (<https://elmerice.elmerfem.org>) and is an ice-flow model built upon a multi-physics FEM package (<https://www.elmerfem.org>) maintained by the partner CSC.

4.11.1 Flagship code/Mini-app(s) initial status

The source-code is available on the official repository (<https://github.com/ElmerCSC/elmerfem>), which also contains the

ELMER/ICE part and is licensed under (L)GPL. There was an attempt to do a POP audit of ELMER some years before ChEESE-2P, which failed because of technical issues of porting the code to the machine for testing. Nevertheless, there are scalability tests of the code available from runs on CSC's supercomputers – which by the multi-physical nature have a large variety (<https://github.com/ElmerCSC/elmer-linsys>). The current challenge we face is to port the code to different GPU platforms.

During the writing of the proposal for ChEESE-2P, we opted to not introduce a mini app. Since there are 140 different modules (with about half of them representing a solver) for a specific physical model, we introduced an option to the cmake building process of ELMER to selectively compile only the necessary input/output modules and basically only provide the ELMER-library (which still is large). Hence, to have a faster compilation time for testing, we can skip ELMER's unnecessary physical models. This vastly reduces the compilation time and helps during development. We have ELMER running on a variety of HPC platforms on several continents. For ChEESE, the most relevant are the EuroHPC platforms LUMI-C and LUMI-G (installation done by CSC) and Leonardo (installation done by CINECA). These two platforms are the main target of the Performance Portability campaign. Aside from this, ELMER(/ICE) is in operation on CSC's national infrastructure, i.e., Mahti as well as Puhti. Internationally, ELMER is installed on Australia's NCI Gadi. Elmer also has been ported to JAMSTEC's Earth-Simulator 4 (AMD-EPYC) and RIKEN's Fugaku (ARM64) in Japan. Earlier publications also indicated that ELMER/ICE was run on Archer2 in UK.

4.11.2 Porting

Porting to LUMI-G involved significant adaptation of the code to compile with the HPE provided software and compiler stack, initially CCE16. After the system update of LUMI to CCE17 it turned out that quite a few regressions in the Fortran compiler prohibited compilation of the code we currently rely on a container provided to us by HPE using the currently unreleased CCE18 to compile ELMER and run it under CCE17). Alternatively, we can use the available GNU GCC stack for production run but thereby taking away the possibility to use OpenMP target offload (not supported by GCC on LUMI). Tests with *clang/clang* (also development versions provided by AMD) showed that currently is not possible to compile ELMER with this suite.

We were working on two main aspects to port ELMER/ICE to run on GPUs of LUMI. On the one hand we introduced OpenMP target offload directives into one solver, as well as an interface to a HIP-based library for solution of sparse matrices, *rocALUTION*, provided by AMD within the ROCm software stack. Both elements are currently tested. Before the campaign, there was already an existing interface to the NVIDIA-provided, CUDA-based linear algebra library, AMGX (<https://github.com/NVIDIA/AMGX>). During ChEESE-2P, a new interface for utilizing AMD GPUs based on ROCm/HIP, *rocALUTION* (<https://github.com/ROCm/rocALUTION>), has been introduced (Fig. 4-11.1).

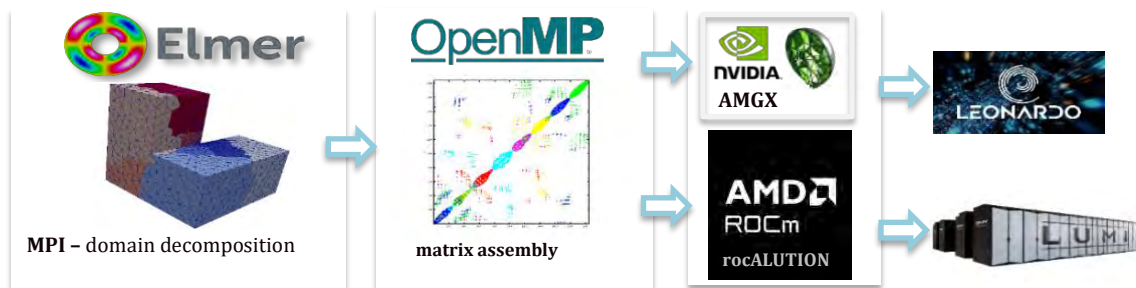


Fig. 4-11.1: Concept of porting and portability for Elmer to/on GPU accelerated EuroHPC systems.

4.11.3 Performance Portability campaign

This means, we can utilize the A100 on Leonardo as well as the Mi250X on LUMI-G and compare the performance of both. Performance testing is based on a benchmark case that can be easily scaled to any problem size and involves a simplified Poisson-solver (<https://github.com/ElmerCSC/elmer-linsys/tree/cheese/Poisson/WinkelUnstructured-ChEESE>). This particular case was also used for the POP audit and (with a to CPU ported version of the solver) will be used in the second audit on LUMI-G. Results, using AMGX on Mahti revealed a good scaling behaviour, with a sweet-spot (scaling + runtime) of one MPI rank/GPU, of which there are 4 in one node (green line in the Fig. 4-11.2)

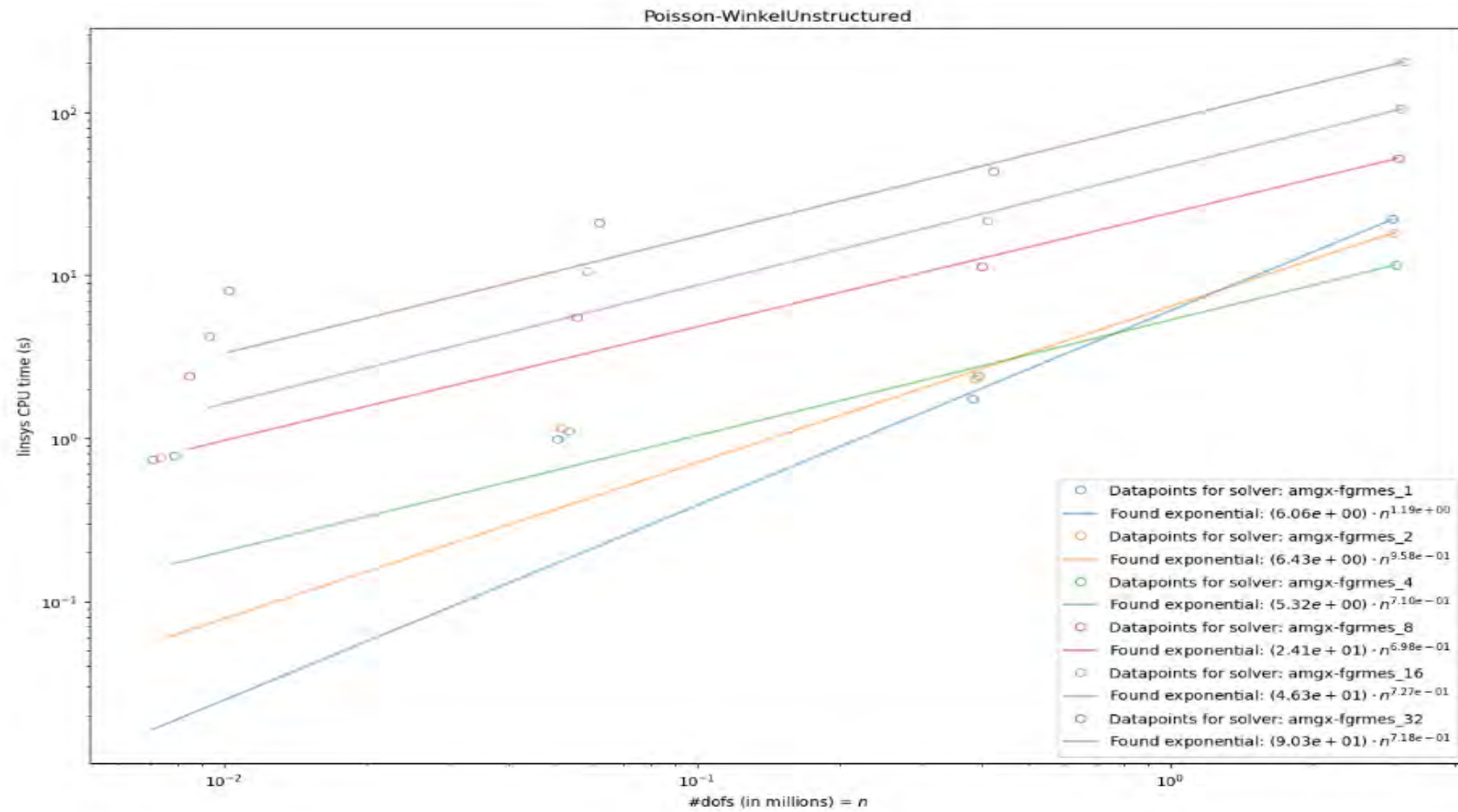


Fig. 4-11.2: Solver timings as function of solved degrees of freedom (DOFs) for different combinations of MPI ranks on a single GPU-node (2xAMD-EPYC + 4xNVIDIA A100). Result taken from <https://github.com/ElmerCSC/elmer-linsys/tree/main/results/Poisson-WinkelUnstructured>:

Different combinations of solvers and pre-conditioners also lead to varying run-times (Fig. 4-11.3):

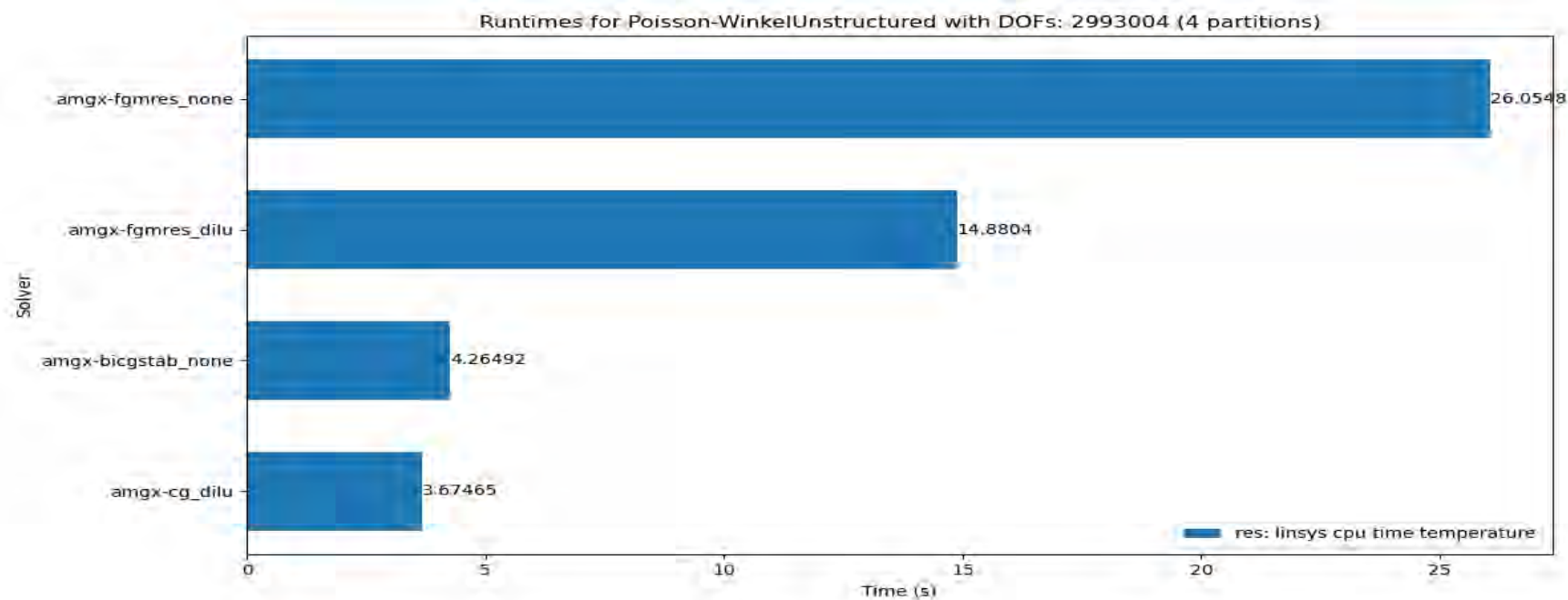


Fig. 4-11.3 Solver timings choosing different solver and pre-processor variations for AMGX GPU solution on a single GPU-node (2xAMD-EPYC 128 cores+ 4xNVIDIA A100). Result taken from <https://github.com/ElmerCSC/elmer-linsys/tree/main/results/Poisson-WinkelUnstructured>:

For the PPC, CINECA were running the benchmark problem on three machines, Leonardo Booster (NVIDIA A100 with AMGx), LUMI-G (AMD MI250X with rocALUTION) and Thea (NVIDIA multi-node Grace-Hopper Evaluation System with NVIDIA GH200 with AMGx). ELMER comes with the possibility to get timings of either the whole *simulation* (Initialization overheads + I/O overheads + matrix assembly + linear system solution), a single *solver-step* (matrix assembly + linear system solution) and a single *linear system* solution step alone. The latter is a good metric for performance comparison of different GPU implementations, as the solution is offloaded to GPUs. The benchmark case can easily be adapted by choice of a particular mesh-level. Increasing the mesh-level by one integer means doubling the nodes (by splitting mesh-edges) in every direction, so about an 8-fold increase in mesh resolution in three-dimensions with respect to the previous mesh level. Tests have been performed on a single computational node, using one MPI rank and one GPU, considering different combinations of CPU threads. For the previously mentioned test-case, we obtain the following timings (mind that these are the 95% confidence intervals of ten runs without any attempt to optimize performance on a particular platform).

Mesh Level 2	MPI/GPU/Threads	simulation	solver-step	Linear system
LEONARDO	1/1/1	5.46±0.03	2.99±0.01	1.68±0.01
LUMI-G	1/1/1	3.45±0.02	1.71±0.01	0.53±0.01
THEA-GH	1/1/1	2.99±0.03	1.26±0.03	0.44±0.03
LEONARDO	1/1/32	4.12±0.03	1.69±0.01	1.62±0.01
LUMI-G	1/1/56	2.58±0.05	0.62±0.01	0.57±0.01
THEA-GH	1/1/72	2.19±0.03	0.38±0.02	0.36±0.02

Mesh Level 3	MPI/GPU/Threads	simulation	solver-step	Linear system
LEONARDO	1/1/1	26.18 sec	11.50 sec	1.70 sec
LUMI-G	1/1/1	22.73 sec	9.72 sec	0.65 sec
THEA-GH	1/1/1	16.88 sec	6.78 sec	0.47 sec
LEONARDO	1/1/32	16.38 sec	2.09 sec	1.68 sec
LUMI-G	1/1/56	14.32 sec	1.02 sec	0.68 sec
THEA-GH	1/1/72	10.57 sec	0.66 sec	0.51 sec

Mesh Level 4	MPI/GPU/Threads	simulation	solver-step	Linear system
LEONARDO	1/1/1	173.70 sec	77.44 sec	2.76 sec
LUMI-G	1/1/1	163.63 sec	71.13 sec	2.03 sec
THEA-GH	1/1/1	124.26 sec	52.62 sec	1.05 sec
LEONARDO	1/1/32	99.95 sec	5.47 sec	2.63 sec
LUMI-G	1/1/56	96.84 sec	4.28 sec	1.96 sec
THEA-GH	1/1/72	71.57 sec	2.01 sec	0.92 sec

Tab. 4-11.1: Performance Portability campaign timings: overall.

The results of the whole simulation are less relevant, as they contain all the I/O overheads that, mainly for the smaller mesh-levels, can take a significant part of the simulation but are less representative of large simulations. The solver-step in relation to linear system timings can give insight on the proportionality between matrix assembly and linear system solution.

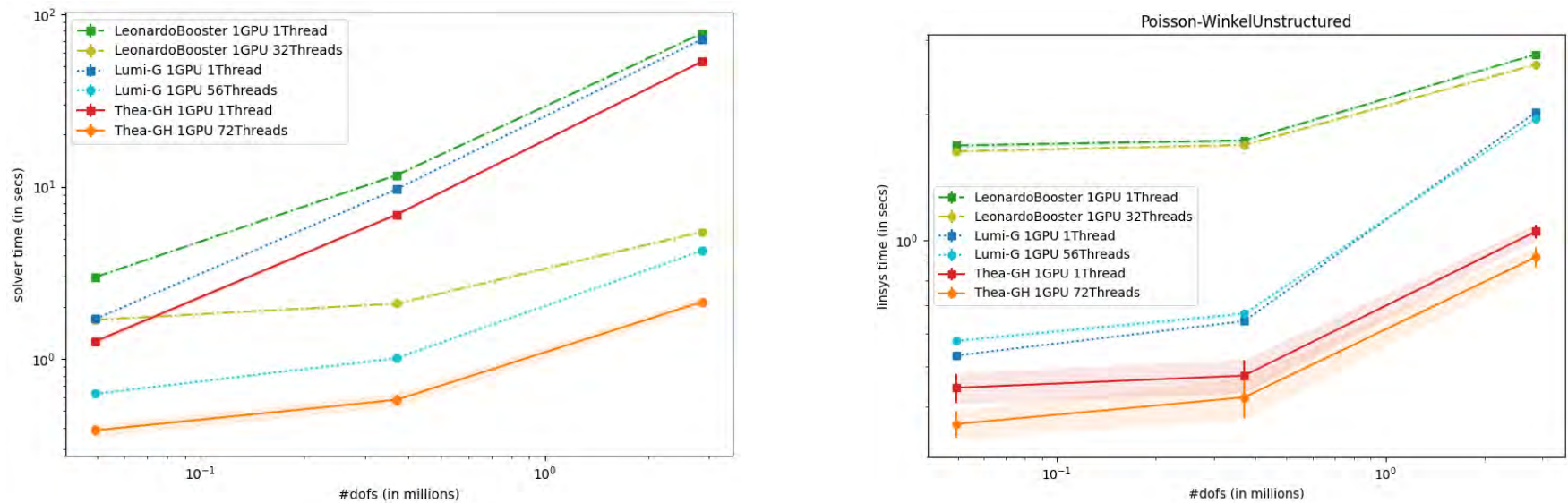


Fig. 4-11.4: Performance Portability campaign timings: Solver time (left) and Linear system (right).

Mind, that AMGX and rocALUTION come with different configurations. The choice here was to use a combination of Conjugate Gradient iterative solver (both on AMGX and rocALUTION) but with different pre-conditioners, namely, DILU (AMGX) and Jacobi (rocALUTION), which should be close enough to provide a basis for comparison.

4.11.4 Code Audit driven improvements

The POP audit within ChEESE-2P was performed on CPU system only (LUMI-C), with a CPU highly optimized code (MPI, OpenMP-SIMD) that did not reveal a significant bottleneck, apart perhaps from a slight load-imbalance induced by the asymmetric domain decomposition of the geometry (which is impossible to avoid for real-world applications, such as glaciers). Our approach was to get a baseline for a second later audit performed on a GPU accelerated machine (LUMI-G) using the accelerators for the solution step of the linear system. Thus, we are not following a strategy to improve the performance of the CPU-only code.

4.11.5 Other improvements

In the context of the ChEESE-2P project and the porting of ELMER/ICE to the AMD GPU's, the ELMER team got support of 3 person-months from HPE, AMD and the Lumi Support Team (LUST) in the framework of a LUMI EuroHPC grant. Part of this project was the participation in the LUMI hackathon in Brussels, where we profiled parts of the code and worked on OpenMP target-offload implementation of the matrix assembly done in the solver of the benchmark as well as improving the interface to rocALUTION (enabling of MPI, i.e., inter-node communication).

4.11.6 Exascale ambition

The activities in the project (including the LUMI EuroHPC grant) should bring us to a level where we can utilize a large number of LUMI-G nodes to compute problems of adequate size, necessary to cope with the challenges imposed, in particular, by SC9.1 in ChEESE-2P. We must resolve a 30 km by 50 km area with expected horizontal mesh resolutions down to about the order of 10 meter. We expect about 10 million of mesh elements, which with 4 nodal degrees of freedom (DOFs) from the incompressible visco-elastic Navier equations make a system of up to 500 million of DOFs to be solved with multiple scenarios to be tested (capacity). That system size exceeds current problems in ELMER/ICE by at least an order of magnitude.

5. Conclusions/Lessons learned

The Tab. 5-1 summarizes the effort spent in the different sections of D2.2 (code by code) and the Tab. 5-2 describe the complete ecosystem (flagship codes and mini apps used in the T2.2 investigation) including FOM and PP metrics:

Flagship code	Porting	PPC	CAD improvements	Other improvements
SeisSol			N.A.	
SPECFEM3D	N.A.			
ExaHyPE				N.A.
Tandem			N.A.	
xSHELLS	N.A.	N.A.	N.A.	N.A.
HySEA				N.A.
FALL3D	N.A.			
OpenPDAC	N.A.	N.A.	N.A.	
LaMEM			N.A.	N.A.
pTatin3D		N.A.		N.A.
ELMER / ICE			N.A.	

Tab. 5-1: effort spent in the different sections of D2.2. Green means that code X participated in the specific task during the reporting period.

Flagship code	Mini apps	GPU flavors	GPUs for PPC	FOM/PP metrics
SeisSol	SeisSol-proxy	CUDA, SYCL, OpenMP target offload	A100, H100, GH200, MI250X, Intel Data Center GPU Max 1550	Max TFLOP/s/Hardware Efficiency (Tab. 4-1.1)
SPECFEM3D	MiniSPECFEM3D	CUDA, HIP	A100, MI250X	Time Loop/Application Efficiency (Tab. 4-2.1)
ExaHyPE	ExaHyPE kernels	CUDA, OpenMP target offload	A100, H100, Intel Data Center GPU Max 1550	Bandwidth/Architectural Efficiency (Tab. 4-3.1)
Tandem	Static mini app	CUDA, HIP	A100, MI250X	TFLOP/s/Architectural Efficiency (Tab.4-4.1)
xSHELLS	N.A.	N.A.	N.A.	N.A.
HySEA	Tsunami-HySEA	CUDA, SYCL	A100, H100, GH200, Intel Data Center GPU Max 1100, Intel Data Center GPU Max 1550	Runtime/ Application Efficiency (Tab. 4-6.1)
FALL3D	FALL3D suite	OpenACC	A100, H100, MI250X	Runtime/Application Efficiency (Tab. 4-7.4)
OpenPDAC	N.A	N.A	N.A.	N.A
LaMEM	MatMult, VecMAXPY (PETSc).	CUDA, Kokkos, HIP	A100, GH200, MI250X	MFLOP/s/Architectural Efficiency (Tab.4-9.1/2/3)
pTatin3D	SpMV MF	CUDA, HIP	N.A.	N.A.
ELMER / ICE	ELMER simplified benchmark	CUDA (AMGX), HIP (Rocalution), OpenMP target offload	A100, GH200, MI250X	Solver time (time-step) (Tab. 4-11.1)

Tab. 5-2: flagship codes, mini apps used, PPC data (when available) in T2.2.

Starting from obtained results, we can assume some conclusion for each specific section:

5.1 Porting

The porting process has proven to be a crucial step to ensure the success of a PPC campaign and, ultimately, to concretely implement the strategy outlined in Task T2.2.

This is particularly true for the "new" codes introduced in ChEESE-2P (which is reasonable, given that the effort invested originated, in most of the cases, from this project).

In general, however, almost all codes underwent a GPU porting phase, although this activity was not formally reported as Porting since it did not result in new codes or mini-apps using GPU-oriented languages beyond those that were already in place at the start of the project.

Several approaches were considered, including SYCL, OpenMP target offload, as well as HIP and CUDA in some cases. Codes based on PETSc began experimenting with Kokkos, thanks to the ability of PETSc to implement multiple backends in a relatively "simpler" way compared to other flagship codes.

5.2 Performance portability campaign

PPC is a milestone in WP2 and the innovative element of D2.2. The methodology appears to be well-defined and solid. 8 out of 10 codes participated in the PPC, which enabled the expansion of the initial ecosystem (flagship code + mini app(s)) to the current one.

GPU-enabled programming paradigms have been extensively explored, including technologies such as CUDA, HIP, as well as Kokkos and OpenMP target offload. Regarding architectures, EuroHPC systems (Leonardo, MareNostrum 5, and LUMI) were used, along with other machines or external software development vehicles (SDVs).

However, some, if not many, PP results are still embryonic or preliminary. The range of software solutions has not yet stabilized, which has occasionally prevented the calculation of PP metrics. Furthermore, the differences between development and production environments across very diverse machines have led to extended campaign timelines, occasionally limiting the scope of what could be achieved with certain flagship codes in the D2.2 timeframe.

5.3 Code audit driven improvements

The difficulty in implementing the recommendations from the code audit is clear, as only 5 out of the 11 flagship codes incorporated plans to enhance performance based on the findings of D2.1.

Several factors contributed to this challenge. Among the others:

1. Some codes underwent the audit using a CPU-focused version, despite the intention to develop a GPU-specific version for D2.2 (ELMER/ICE as an example)
2. The level of effort provided by the partners, as well as the support from POP3 contributors (BSC and HLRS), was partly insufficient to effectively address this phase in a more comprehensive manner.

However, for the codes that succeeded in acting on the audit's recommendations, the results were highly significant and impressive. Notable examples include SPECfem3D and FALL3D, which demonstrated substantial improvements, but this is true for the rest of the flagship codes. Looking ahead, corrective measures should be introduced, particularly in preparation to D2.4, to ensure stronger alignment and support for this critical aspect of the project (see Lessons Learned section).

5.4 Other improvements

Half of the codes have experienced with this part of the activities. By its nature, this part stems from existing needs (particularly from code owners) already included in the scope and that do not require further input.

A lot of emphasis has been placed on the algorithmic aspects or issues related to the complete workflow (from the code's perspective) and/or on currently experimental issues that could potentially boost performance (in some favourable cases).

Only in some cases (ELMER, for example) this part involved the use of GPU-oriented tools or MPI-based approaches.

5.5 Lessons learned

Finally, Lessons learned from D2.2 are highlighted in the following final table:

Category	Issue Name	Problem/Success	Impact	Recommendations
Porting	Increasing the mini apps ecosystem	Although almost every code has investigated Porting, it has been challenging to explore all possible strategies and ultimately prioritize those with the greatest chances of being performance portable (e.g., using tools like Kokkos and/or OpenMP target offload). Additionally, the large number of flagship codes has naturally limited the potential.	The impact was reduced, but for some code it was not possible to produce relevant PP metrics.	Increase the number of mini apps using performance portable layers (Kokkos, OpenMP target offload). Ensure adequate support and provide specific training.
PPC	Improving the PPC outcomes	The PPC in D2.2 involved almost all the codes, the main EuroHPC architectures (and other machines), and produced some interesting results in terms of scaling and PP metrics computation. However, although the methodology is consistent, its practical implementation has been somewhat cumbersome. Moreover, considering both EuroHPC and SDV hardware required a proportionally greater effort than the planned one. Additionally, the distinction between application efficiency and architectural efficiency was proved challenging for many flagship codes on a given architecture.	The impact was reduced, but for some code it was not possible to produce relevant PPC outcomes.	Increase the number of mini apps using performance portable layers (Kokkos, OpenMP target offload). Ensure adequate support and provide specific training. Simplify the complex procedure for producing PPC metrics. Disseminate PPC methodology and outcomes internally in the project.
CAD improvements	Effectively use (most of) the outcomes from code audit	The code audit was a joint effort (well-established from ChEESE) between WP2 and POP3 partners (BSC and HLRS). Although all flagship codes participated in the audit campaign, it was not easy to work on the recommendations provided by POP3, despite the support of POP3 itself. Moreover, some code focused their work in D2.2 on porting to	The impact was medium. For some code it was not possible to use all the outcomes from D2.1.	Try not to synchronize the second code audit step, so that the code audit of a given flagship code can proceed independently. This should guarantee more time and effort to accomplish and produce more concrete and useful outcomes to be used in T2.3.

		GPUs, making the initial code audit and its recommendations less useful. Finally, the POP metrics, which are useful for comparing performances in D2.1 and D2.3, are not particularly helpful in addressing the bottlenecks of a flagship code.		Ensure adequate support from POP3 also in this step (exploiting POP3 services) or provide specific training (beyond what is already expected from POP3 channels). Simplify the complex procedure for producing an assessment (at least on EuroHPC) and in cooperation with WP4 for the part related to assessment tools (selection and use). Disseminate POP3 methodology and tools internally in the project. Use TALP and TALP-pages for a quick-and-dirty evaluation of performance.
--	--	---	--	--

Appendix

List of the hardware used in the overall activities (main platforms)

Machine	GPU	Notes
Adastra	AMD Instinct MI250X	https://dci.dci-gitlab.cines.fr/webextranet/architecture/index.html
Claix	NVIDIA H100	https://top500.org/system/180295/
Leonardo (booster)	NVIDIA A100 SXM4	https://www.top500.org/system/180128/
LUMI-G	AMD Instinct MI250X	https://www.top500.org/system/180048/
MareNostrum 5 (ACC)	NVIDIA H100	https://www.top500.org/system/180238/
SuperMUC-NG Phase 2	Intel Data Center GPU Max 1550	https://doku.lrz.de/hardware-of-supermuc-ng-phase-2-222891050.html
Thea	NVIDIA GH200 "superchip"	N.A.
Unknown system	Intel Data Center GPU Max 1100	N.A.
Vista	NVIDIA GH200 "superchip"	https://tacc.utexas.edu/systems/vista/

ⁱ The answer is clearly 42