D3.1 Application bottlenecks and optimization opportunities on heterogeneous components

Version 1.0

Documentation Information

Contract Number	9555558
Project Website	www.eFlows4HPC.eu
Contractual Deadline	30.11.2021
Dissemination Level	PU
Nature	R
Author	Universitat Politècnica de València (UPV)
Contributors	Rosa M Badia (BSC), Enrique S. Quintana-Ortí (UPV), Domenico Talia (DtoK), José Flich (UPV), Eugenio Cesario (DtoK), Alessandro D'Anca (CMCC)
Reviewer	Gabriele Accarino (CMCC)
Keywords	Bottlenecks, compute, communication, storage

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

Change Log

Version	Description Change
V0.1	First draft of the deliverable
V0.2	Major input from meetings from pillar partners
V0.3	Applied corrections as suggested during internal review
V1.0	Version formatted for submission

Table of Contents

1. Executive Summary

This deliverable identifies potential bottlenecks for the Applications (Pillars) targeted in the eFlows4HPC project. The actual objective of this effort is to accelerate the execution of the corresponding workflows by developing high performance realizations of selected numerical kernels as well as storage and communication alternatives to tackle the identified bottlenecks. The focus of the work in WP3 is on the benefits that new architectures (in the form of communication and storage technologies and/or heterogeneous architectures) may bring to the acceleration of key components of the workflows and, in consequence, the mitigation of the bottlenecks.

Several meetings and working sessions have been held with the Pillar partners. In addition, collected reports, some of them POP CoE audits, for specific frameworks/models used within the workflows have been obtained and carefully analysed. From all these meetings and reports, and from the Workflow pillar descriptions (Deliverables D4.1, D4.2, D5.1, D5.2, D6.1, and D6.2), we have structured all information and compiled this deliverable.

In addition to the identification of potential bottlenecks, we also provide some hints on the optimization opportunities to alleviate these bottlenecks. This is done from the perspective of using the additional architectures targeted in the project, namely GPUs (graphics processing units), FPGAs (field programmable gate arrays) and EPI (European Processor Initiative) as well storage and interconnect technologies.

This report describes the activity performed in T3.1. The report sets up the baseline for the work of the remaining tasks of WP3.

2. Introduction

The different workflow applications, described in WP4, WP5 and WP6 (for Pillars I, II and III, respectively), and exercised in the project, present a considerable complexity. In particular, all of them are composed of many processes, some of them running in parallel while others are executed sequentially. Some processes may even run in the same workflow on different machines with distinct configurations. Therefore, searching for bottlenecks in the entire project workflows requires careful analyses, together with a deep study on the implications of the three main performance components: computation, communication, and storage.

In order to identify the potential compute-specific bottlenecks in the Pillars, we first define the concept of a (computational) *kernel*. This represents a concrete algorithm that implements a specific functionality in a computer system. The same kernel may have different realisations depending on the target system where it is actually implemented, for instance a CPU (central processing unit), a GPU or even an FPGA.

Different types of bottlenecks may appear in a workflow. Depending on the type of performance component that causes the bottleneck, we can classify these bottlenecks as:

- Compute-specific
- Communication-specific
- Storage-specific

For compute-specific bottlenecks, we identify the kernels that produce them as the kernels that dominate the compute time of a running workflow application. We expect that improving those kernels by targeting new compute-specific resources (e.g., a high-end GPU or an FPGA) reduces the execution time and/or the energy consumption of that kernel, and exerts a significant impact on the final efficiency of the workflow. Additionally, we may find compute-specific bottlenecks not related to a specific kernel but to the whole process. Indeed, hundreds or thousands of simulations may be required posing a compute-specific bottleneck per se to the workflow.

For communication-specific bottlenecks, we identify data transfers between workflow modules as indicated in the deliverables describing the workflows (D4.1, D5.1, and D6.1). Those communication transfers may be also affected when communication occurs off premise. In that case, we need to consider both the communication intensity and the bandwidth of the interconnection network. Communication-specific bottlenecks are also identified within an HPC (High Performance Computing) system, mainly when running the simulation frameworks of models for a specific workflow. MPI communication is the preferred communication protocol leveraged by the HPC components for most workflows and we may track communication-specific bottlenecks down to specific communication routines in the MPI standard.

Finally, storage-specific bottlenecks may arise when large amounts of data stored in local disks need to be accessed. In those situations, a storage-specific technology (e.g., NVRAM disks) may help in reducing the data access time.

The work carried in Task 3.1 (which is the basis for this deliverable) included many meetings with the Pillar partners, with lively discussions on their workflows and current knowledge of the potential bottlenecks. The meetings were also complemented with previous (and recent) analysis reports provided by the Pillar partners, some of them POP CoE Audits, with accurate bottleneck identifications for the specific key components used in the workflows.

For the identified bottlenecks, during the project execution we expect to work on solutions to attenuate and reduce the impact of the bottleneck on the workflow. These solutions come from the combination and adoption of alternative technologies that may be used in the project.

Compute bottlenecks will be labelled through this document as CP.X.Y where X is the Pillar (I, II, or III) and Y is the bottleneck number. The same way, communication bottlenecks will be labelled as CM.X.Y and storage bottlenecks as ST.X.Y.

The rest of the document is structured as follows. Sections 3, 4, 5 and 6 are devoted to the three project Pillars. For each Pillar, in the corresponding section we first identify the compute-, communication- and storage-specific bottlenecks, and then draft possible optimizations that may alleviate those bottlenecks in the context of the Pillars. Then, in Section 7 we provide a summary of the insights gained from the present study and briefly discuss how the new architectures and technologies may be used to help alleviate the identified bottlenecks. Finally, in Section 8 we close the report with a summary of the main findings.

3. Pillar I: Reduced Order Models

The goal of Pillar I is to develop an integrated workflow for the realization of reduced order models (ROM), from their inception to their deployment. The workflow general overview is depicted in Figure I. This Pillar targets the simulation of complex engineering problems, with high computational requirements, by developing a ROM that provides an accurate representation of the original system, with a much lower computational cost, which can be then utilized in subsequent stages of the workflow.

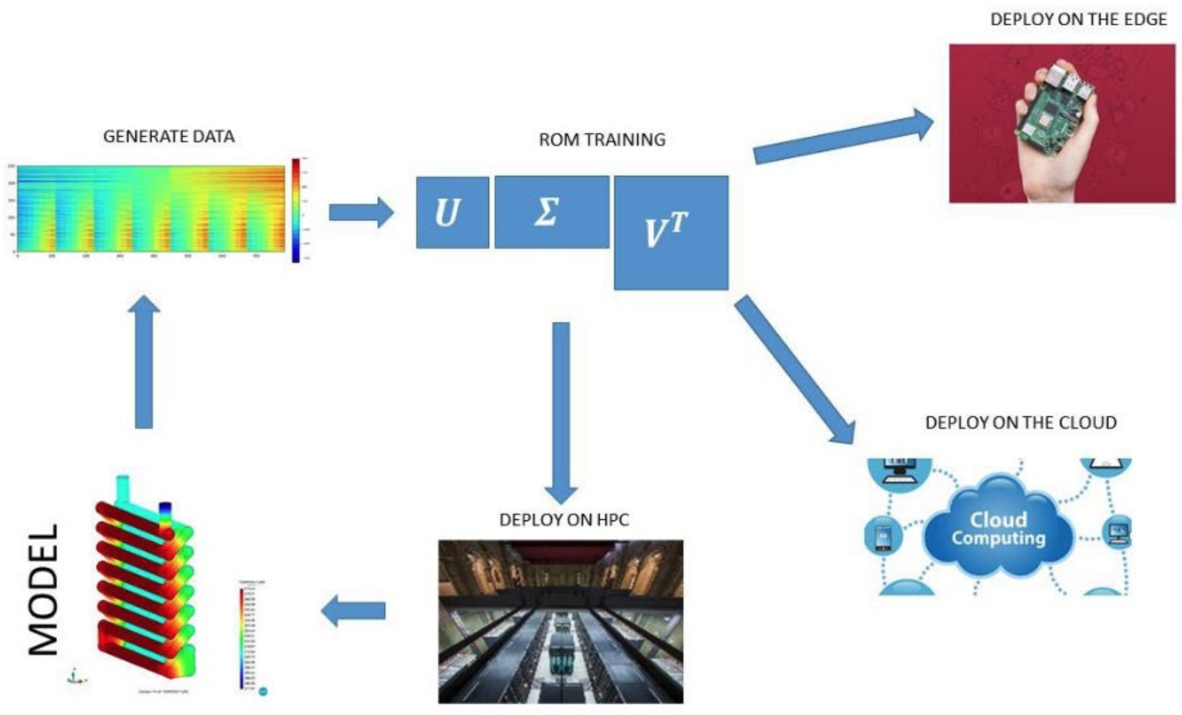

Figure I. General overview of the ROM workflow.

As described in deliverable D4.1, the workflow for Pillar I consists of 5 "stages":

1. Generation of the input data.
2. Data extraction.
3. Hyper-reduction.
4. Validation.
5. Deployment.

Deliverable D4.1 identified stages 1 and 2 as those with the highest computational, communication and storage demands, with stage 1 dominating the entire cost because of the need of performing a large number of (mostly independent) simulations. Stage 2 involves the extraction of data via the Singular Value Decomposition (SVD) algorithm, which implies a large level of concurrency and data transfers.

Although a large-scale SVD is also required in Stage 3, the data generation involved in that stage already operates on a ROM, which should significantly reduce the cost of this process. Stages 4 and 5 operate on a hyper-reduced model, further diminishing the cost of these stages. Therefore, in the following subsections we focus on the first two stages, identifying their main bottlenecks, and quantifying their contribution to the execution time of the workflow.

3.1 Generation of the input data (stage 1)

This stage is tackled in the workflow using KRATOS,[1] an open source framework for the implementation of numerical methods for scientific and engineering problems. The baseline problem that underlies KRATOS, when used to generate the input data for Pillar I, boils down to the solution of a large number of simulations (in the order of hundreds), covering the most relevant scenarios. Each simulation typically represents a time dependent process (with more than 200 time steps). The problems to be described are nonlinear, and require an iterative solution procedure (typically around 3 iterations per time step).

The linear systems appearing in this application are sparse, and present no special structure (e.g., band, symmetry, etc.) other than a symmetry in the underlying graph, nor property (positive definiteness, etc.). In the current realization of stage 1 for the workflow underlying Pillar I, these linear systems are solved using an Algebraic Multigrid (AMG) preconditioner in combination with a Krylov subspace-based iterative solver. Specifically, the Pillar stack employs the AMGCL software package[2] for this purpose. This numerical tool provides backends for multicore processors as well as GPUs on top of OpenCL, CUDA and OpenMP.

To acquire a general understanding of the dimensions of the problem, we performed a preliminary profiling of the AMGCL backend for CPUs, using an Intel(R) Core(TM) i7-1165G7 multicore processor. The selected testcase corresponds to a Navier Stokes problem coming from the discretization of the fluid domain of an electrical motor cooling simulation (selected testcase for the demo of Pillar I). This particular sparse linear system presents about 800.000 unknowns. The GMRES method in AMGCL, combined with an ILU0 preconditioner in the same package, was applied to the solution of the linear system. The analysis reported that the memory footprint for the solver was 609.33 MB and that of the preconditioner was 1.61 GB. The iteration was stopped when the relative residual of the solution was below 1.0e-9, which required 26 iterations. The results from this analysis showed that a significant part of the execution time of the standalone solver (65.79%) was spent reading the data from disk. In a real scenario, though, the data for the linear systems come from a prior stage and this time is therefore discarded for the following analysis. The second and third phases that dominated the execution time were the iteration solve (25.85%) and preconditioner setup (8.35%), respectively. Considering only these two phases, the execution time varied between 85.24 seconds when using a single core, 48.26 seconds for 4 cores, and 35.89 seconds when the full socket (i.e., 8 cores) was employed. The speedup of the preconditioner setup was practically nonexistent: 1.05 and 1.18 when using 4 and 8 cores, respectively. The iterative solve reported a mild parallel scalability: 2.27 and 3.50 for 4 and 8 cores, respectively.

Identified bottlenecks

[CP.I.1] Delving further into the software stack, depending on the convergence rate of the iterative solver, in general the computational cost of stage 1 in the workflow is dominated either by the construction of the AMG preconditioner or the Krylov subspace-based iterative solver. In the former case, depending on the type of preconditioner that is applied, special attention must be paid to the kernel for the *sparse matrix-matrix multiplication (SpMM) and sparse triangular solve (SpTrsv)*. For the latter case, we need to take into account that the selected iterative process in Pillar I is a variant of the conventional GMRES or BiCGStab Krylov subspace methods. In

[1] https://www.cimne.com/kratos/
[2] https://github.com/ddemidov/amgcl/

consequence, the critical kernel in this case may be the *sparse matrix-vector product (SpMV) or the orthogonalization process involved in the GMRES*. For GMRES, *the kernel for the general matrix-vector product (GeMV)* that is necessary to estimate the search direction of the method can also play an important role from the computational perspective.

[CP.I.2] From a computational point of view, the global cost is dominated by a sparse linear system of equations, representing the Jacobian of the FEM problem to be minimized. In the case of performing 100 simulations, each with 200 time steps and with 3 nonlinear iterations, one would need to solve 600 times the linear system of equations per simulation (so a total of 60,000 times). We remark that each simulation is completely independent from the others and, therefore, the problem is embarrassingly parallel at that level.

Potential optimizations

The (compute-specific) kernels identified in the first stage (generation of the input data) are memory-bound. This implies that any optimization strategy needs to consider very carefully the cost of moving the data across the memory hierarchy of the computer. Communication-reduction techniques for this type of kernels, dealing with sparse matrices, tackle this problem by a combination of the following: 1) designing specialized data structures to reduce the overhead of the indexing information and/or improve data locality; 2) combining different precisions for the floating point data; and 3) eliminating synchronization points due to collective reductions (though some of these options may come at the cost of numerical stability and impact the convergence rate of the iterative solver.) Significant gains can also be attained via the use of alternative preconditioners which present a better fit to the target problem. In the project, the use of FPGAs paves the road to the exploration of customized floating point formats to store the data in memory and perform the floating point arithmetic.

3.2 Data extraction (stage 2)

Identified bottlenecks

[CP.I.3] The baseline realization of this stage relies on the SVD to compress the original system model into a ROM, where the accuracy of the representation is adjusted via the singular values, which are used to truncate the decomposition to a certain point. In the current implementation, the use of a randomized variant of this decomposition [Mar20] significantly reduces the computational, communication and storage costs of this operation. In some detail, the randomized SVD is obtained via 1) a couple of large-scale general matrix-matrix multiplications (GeMMs), 2) the QR factorization of a tall-and-skinny matrix, and 3) the SVD of a small, "squarish" matrix. Taking into account the dimensions of the operands that participate in each one of these computations, the computational *kernel that dominates this stage is the GeMM* followed, to a much lower extent, by the QR factorization. The contribution of the final SVD is negligible; see Table I.

[CM.I.1] As a starting point for the identification of the bottlenecks in this stage of Pillar I, we implemented[3] a prototype version of the randomized SVD as a PyCOMPSs script that builds upon the dislib[4] methods. The initial strategy designed for this purpose was based on the distributed

3 The evaluation of stage 2 of the ROM Pillar was performed on the MareNostrum 4 (MN4) Supercomputer. Each node of this platform has two Intel Xeon Platinum 8160 (24 cores at 2,1 GHz each) and 96GB of main memory. More details of MN4 can be found in https://www.bsc.es/marenostrum/.
4 See D1.1 for a description of dislib and PyCOMPSs.

array (ds-array), which is the basic dislib data structure. A ds-array is a matrix divided into blocks that are stored in a distributed way in a set of nodes of a computational cluster. Each block of a ds-array is a NumPy array or a SciPy CSR (Compressed Sparse Row) matrix, depending on the type of data used to create the ds-array. The script invokes a few dislib methods, following the order indicated in Table I: GeMM, QR factorization, transpose, and SVD. The script was first tested with small size matrices for correctness, and the size was then increased to assess the experimental costs of the components on realistic values for Pillar I. This initial evaluation revealed a performance drop when operating with an input matrix A of one million rows by 5000 columns, and a matrix □ of size 5000 by 110. The main reason for this performance decline was the dislib method for the QR factorization, which required its input ds-array to be organized into square blocks. Given the size of the input matrix, this implied that the largest block size was 110 by 110. In consequence, PyCOMPss generated a considerable number of tiny tasks, yielding a significant overhead due to task management.

Table I. Operations in the computation of the randomized SVD in stage 2 of the ROM Pillar. In the problems to be solved in this Pillar, the dimension n *is in the order of millions to tens of millions (*n *~ $O(10^6\text{-}10^7)$) while* m *is in the order of few thousands (*m *~ $O(10^3)$). The third problem dimension,* k*, needs to be dynamically adjusted during the computation process but it is usually in the range of a few hundreds.*

Computation	**Kernel**	**Theoretical cost (in floating point operations)**
1. Generate random □; compute Y=A □	GeMM	*2nmk*
2. Compute truncated QR factorization Y = QR	QR	$O(nk^2)$
3. Compute B = Q^T A	GeMM	*2nmk*
4. Compute the SVD B=U_BSV^T	SVD	$O(mk^2)$
5. Compute the left singular vectors U = QU_B	GeMM	$O(nk^2)$

As part of our initial effort to identify the actual bottlenecks in this stage, we refined the prototype implementation for the randomized SVD in order to leverage the distributed array and the dislib methods in steps 1, 3 and 5 (including the computation of the transpose) and to use local operations for steps 2 and 4. In order to invoke the local operations, in this second version the distributed data was gathered in one cluster node, the corresponding method (QR or SVD) was invoked locally, and the data was again converted to a ds-array for the subsequent step. While this fan-in/fan-out of data introduced some communication overhead, the strategy proved to be much more efficient, and we were able to compute the randomized SVD of matrices with one million rows and 5000 columns in 1-2 minutes; and for matrices with 10 million rows and 5000 columns in less than 4 minutes.

As mentioned previously, we observed some overhead sources in this stage when gathering and scattering data from multiple computing nodes in the randomized SVD script. These overheads occur when changing from the distributed format of the ds-array to a centralized format (a single NumPy array). In its current implementation, the gathering implies a number of data transfers from multiple nodes to one node, followed by a sequential NumPy block operation to convert the blocked list into a single NumPy array. This operation is currently encapsulated in the "collect" method of the ds-array.

[ST.I.1] In order to exchange data in memory between nodes, PyCOMPSs serializes the tasks' output objects into files. The profiling performed with the initial tests of the randomized SVD script allowed us to detect an issue due to the data serializations that PyCOMPSs performs. In particular, we observed an increase in the serialization times when generating random ds-arrays using as temporary storage the local disk in the nodes. This problem did not appear when using the shared file system.

Potential optimizations

The profiling of the current script exposes some bottlenecks in step 2, both for the gathering/distribution of data and the QR itself, and in the transposition that appears prior to step 3. We have identified the following potential optimizations to alleviate these bottlenecks (only computational bottlenecks are listed below):

- Optimization of the distributed QR by implementing a version for tall-and-skinny matrices in dislib.
- Extension of the dislib routine for matrix multiplication to support transposed matrix operands.
- Reformulation of the gathering as a reduction operation, enabling dislib to perform the block operation in parallel. We foresee a solution based on the reduction decorator of PyCOMPSs (@reduction) that performs the corresponding operation as a set of tasks, first operating at node level and later between nodes, reducing the number of communications and exploiting the locality, while at the same time the operation is performed in parallel.

At this point we want to mention that a more challenging optimization is to replace the complete calculation of stage 2 (data extraction) with a module based on deep neural networks, possibly via autoencoders. The evaluation of the potential of this approach is still in a germinal stage, to the point where it is too early to identify potential bottlenecks. It is however interesting to remark that, in many cases, we use the SVD as a first step of reduction, and we then leverage the data projected onto the SVD basis to train the autoencoder. This implies that the "heavy lift" in the autoencoder reduction is still done with the linear SVD kernel allowing to take advantage of the optimization described just above.

4. Pillar II: Dynamic and adaptive workflows for climate modelling

The use case defined in Pillar II is divided into two workflows (see Figure II). The first workflow is the ESM (Earth System Model) Dynamic (AI-assisted) workflow whereas the second one is the Feature Extraction and Statistical Analysis workflow. In some detail, the first one leverages specific software packages from the climate science domain, in particular OpenIFS and FESOM2, in order to conduct climate simulations. In consequence, the identification of bottlenecks for this workflow is focused on OpenIFS and FESOM2, as they represent the most compute-intensive parts.

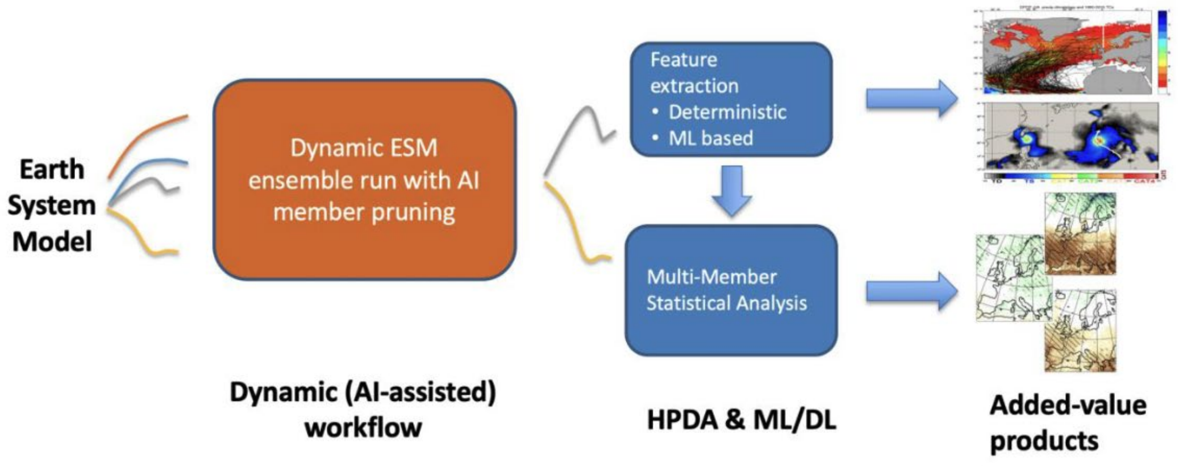
Earth System Model
Dynamic ESM ensemble run with AI member pruning
Dynamic (AI-assisted) workflow
Feature extraction
• Deterministic
• ML based
Multi-Member Statistical Analysis
HPDA & ML/DL
Added-value products

Figure II. General view of the dynamic and adaptive workflow for climate modelling.

The second workflow defines two use cases with input sources reflecting two distinct application scenarios. The first use case represents the base scenario where extraction and analysis of Tropical Cyclone (TC) detection and tracking information are applied on the output of a single climate model, specifically CMCC-CM3. This model is defined in the Community Earth System Model (CESM) project, with the CESM ocean component replaced by the NEMO physical core, version 4.0. The second scenario considers large volumes of data from centennial CMIP6 products at the highest horizontal resolution available and with a temporal frequency of at least 6 hours. In summary, CMCC-CM3, NEMO, DA, and TC are used in this workflow.

In the following, we discuss the bottlenecks identified in each workflow.

4.1 ESM Dynamic (AI-assisted) Workflow

4.1.1 OpenIFS

OpenIFS is the open access to IFS (Integrated Forecasting System) specifically designed for research and teaching. OpenIFS provides a numerical weather prediction tool that operates with medium range to seasonal timescales. OpenIFS is written in Fortran with some small parts of the code written in C. The software leverages MPI and OpenMP and currently there is no support for GPUs.

Identified Bottlenecks

OpenIFS was recently analyzed in [OPENIFS-BSC]. We next summarize the main findings from that report:

- **[CP.II.1]** According to the general profiling, OpenIFS presents unbalanced computations in the following modules: physical calculations, first block of transposition/transformations, and gnorm/spnorm calculations.
- **[CM.II.1]** OpenIFS also exhibits unbalanced collective communications, with a large number of isend/recv, irecv+wait_any calls.
- **[CM.II.2]** In addition, the report identifies a synchronization point where all the processes need to agree on whether or not to continue the simulation (sigcheck).

OpenIFS can be combined with NEMO. In this case the report identifies two additional bottlenecks:

- **[CM.II.3]** A big number of send/recv calls with small useful duration phases.
- **[CM.II.4]** A large number of Allreductions in the PCGSolv function.

Potential optimizations

The following potential optimizations are proposed in the report:

- Introduce asynchronicity in the calculation of the sigcheck signal, overlapping the execution of the current iteration with a broadcast of the signal from the previous one.
- Introduce asynchronicity in the norm calculations, overlapping the norm calculation corresponding to one iteration with the computations for the next one.
- For the particular case of NEMO, the number of allreductions in PCGSolv can be diminished by relaxing the rate of convergence check in addition to using a SOR (successive over-relaxation) method.

4.1.2 FESOM2

FESOM2 (Finite-volumE Sea ice-Ocean Model) is a multi-resolution sea ice-ocean model that solves the motion equations for unstructured meshes. The performance of FESOM2 has been analyzed in different studies, including some preliminary studies between BSC-ES and ECMWF. Here we first focus on the results from the report by the Earth model performance analysis group at BSC-ES [REP-FESOM2].

Identified bottlenecks

[CM.II.5] The FESOM2 code is divided into three phases: init, init2 and iterative process. The init2 phase does not scale with the number of processors, mainly due to irregularities in the communications that introduce serialization. Indeed, using more processors in this phase increases its execution time.

[CM.II.6] Multiple iterations are performed in the iterative process phase. Each iteration consists of an Ocean (sub)phase and an Ice (sub)phase. There is a large collective communication that consumes 5% of the time and does not scale when a larger number of processors is used.

[CM.II.7] In the Ocean phase there is a load imbalance since some subdomains contain more layers than others. In addition, the Ice phase has a large number of fine grain MPI communications. This represents a communication bottleneck since computation is serialized and asynchronous communication is not possible.

An evaluation using the PAPI (Performance Application Programming Interface) tool revealed the following additional data. The instantaneous parallelism of the time-step phase comprises, on average, 36% of the time. This means that only 36% of the MPI parallel resources are doing useful work (e.g. not waiting for communication). The load imbalance of the ocean calculation produces a reduction of the parallelism when communication is performed. Also, the low granularity of computation between communications affects the Ice phase with low parallelism. This results in the useful IPC (instructions per cycle) rate being in general lower than IPC. Indeed, the performance of the Ice areas is affected dramatically, while the ocean areas experience an impact when close to the communications.

These scalability problems are also analysed in [Koldunov19]. The study in that work concludes that FESOM2 suffers from parallel scalability issues as bottlenecks arise from the saturation of the parallel communication after the number of mesh partitions becomes smaller than a certain threshold of surface vertices per compute core, depending on the model and on the hardware employed.

The components of the ocean circulation models responsible for limiting scalability have been identified to be the solver for the external (barotropic) mode and the sea-ice model. Both represent 2D stiff parts of the solution algorithm and require either linear solvers (usually iterative) or explicit pseudo-time-stepping with very small time steps. Both approaches are not particularly computationally expensive but introduce numerous exchanges of 2D halos per time step of the ocean model. Therefore, the bottlenecks are due to communication and not to computation. In addition, current CPU architectures appear to be well suited for nearly all 3D computational parts of FESOM2.

[CP.II.2] Currently FESOM2 runs on CPU. Porting other compute-intensive kernels beyond tracer advection to GPUs, EPI and even FPGAs may improve performance of the workflow.

[CPI.II.3] One important aspect of this workflow is the high number of simulations that need to be performed in the HPC system. This entangles a computation-specific bottleneck.

Potential optimizations

From the two analyses of FESOM we can derive the following possible improvements for the mitigation of bottlenecks. These improvements point in the direction of increased memory bandwidth, lower communication latency, and more efficient file systems. Therefore, it is vital to choose the “optimal” hardware. Suboptimal scaling of the sea ice combined with a sequential arrangement of sea-ice and ocean steps results in an inefficient utilization of the computational resources and indicates a clear direction for improvement. This, together with a better, scalable, parallel I/O, is the direction for future model code development to enable high-resolution climate simulations with reasonable throughputs.

The following recommendations are listed in [REP-FESOM2]:

- Review the broadcast communication to avoid the irregular pattern that produces load imbalance in the communications (MPI study).
- Review initialization algorithm to avoid the irregular pattern which produces load imbalance in the communications (Init2 study).
- Evaluate the possibility of OpenMP to avoid serialization, the reduction of the number of MPI communications and low computation granularity due to the several communications during the Ice calculation (MPI study).
- Evaluate the possibility of the reduction of the MPI communications during the Ice calculation if OpenMP is not a possibility.
- Review the domain decomposition algorithm to improve the load balance in the ocean calculations (MPI study).
- Since the bottlenecks presented in the previous points result in low parallelism of the model execution, the computational performance of the model is affected, for example, reducing the IPC efficiency (PAPI counter study). In case any of the suggested optimizations explained before is introduced, the parallelism (and subsequently the computational performance) will be improved at the same time.

- Review the locality of the two areas with low IPC and high cache misses to improve the performance of a computation phase (PAPI counter study).
- Review the dependencies in the calculation loops for the area of calculation with low VEC (PAPI counter study).

As commented above, One important aspect of this workflow is the high number of simulations that need to be performed in the HPC system. This entangles a computation-specific bottleneck. One approach envisioned in the project to tackle this issue consists in developing components or functionality in order to enhance ensemble members simulation runs with the capacity to prune members that do not add useful information to the whole simulation. The idea is to make a more efficient use of the computational and storage resources by performing a smart (AI-driven) pruning of ensemble members (and releasing resources accordingly) at runtime. Based on this idea, certain Neural Network (NN) computational kernels, such as GeMM, convolutions (CONV), and activation functions may become potential compute-specific bottlenecks that could be addressed within the project.

4.2 Feature Extraction and Statistical Analysis Workflow

4.2.1 NEMO

NEMO stands for "Nucleus for European Modelling of the Ocean" and is a modelling framework for research and forecasting services in ocean and climate sciences. The NEMO ocean model has three major components (also known as core engines):

- NEMO-OCE models the ocean dynamics and solves the primitive equations.
- NEMO-ICE models sea-ice dynamics, brine inclusions, and subgrid-scale thickness variations.
- NEMO-TOP models the online and offline oceanic trace transport and biogeochemical processes.

The NEMO model is a critical computing component of the Earth System Model (ESM) workflow foreseen in the context of Pillar II, as it represents one of the members included in the CMCC-CM3 model. NEMO is supported by a large Community while several optimizations are being developed in parallel. However, in order to discuss the bottlenecks in this component, we will stay with the official release of NEMO (v4.0). At this point, we note that NEMO has been previously analysed in other projects, such as IMMERSE[5], IS-ENES3 [ISENES-NEMO], PRACE [PRACE-NEMO], and by the Performance Optimisation and Productivity Centre of Excellence in HPC (POP) [POP-NEMO]. In this section of the deliverable, we summarize the results in these documents that are of interest to the eFlows4HPC project.

The baseline version of NEMO is single-threaded and uses MPI for communication. From this starting point, several optimizations have been recently added to NEMO, following the analyses and optimization efforts performed in [ISENES-NEMO] and [POP-NEMO]. The first optimization deals with single core performance. In this optimization, the computation is divided to obtain one MPI domain per node, and each domain is further divided into tiles. The dimension of these tiles

[5] https://immerse-ocean.eu/

is determined according to the node cache geometry in order to avoid data eviction from the processor cache hierarchy, yielding a better exploitation of data locality.

NEMO v4.0.2 has been also profiled and analysed in MareNostrum4, and the results are collected in the POP report [POP-NEMO]. That work provided insights on the communication overheads that constrain NEMO's efficiency. In particular, the north fold was identified as one of the main constraints to the model scalability. The report proposed acting on the granularity of the dynamic solver as a way to reduce idle periods during the execution of NEMO.

The analysis of the communication overheads has also been addressed in [ISENES-NEMO]. Two optimizations were applied there. First, point-to-point primitives, used in NEMO routines to update the halo region before performing computation on the generic point using values of its neighbours, were replaced by MPI3 standard collective communications. Second, the frequency of exchanges was reduced by increasing the dimension of the halo region.

Identified bottlenecks

[ST.II.1] For the NEMO model, there is one clear potential bottleneck in the communication infrastructure. The XIOS library (XML-IO-Server), used for I/O, stores the output of diagnostics and other data produced by climate component codes into files that are then passed to temporal and spatial post-processing modules that operate on this data. XIOS is extensively used in NEMO and is based on the server concept where I/O tasks are leveraged and performed asynchronously. Thus, the diagnostics in NEMO were offloaded from the model and run in parallel with ocean dynamics. Efforts in this direction are underway in order to port these diagnostics to the GPU, whereas the rest of NEMO runs completely in CPU.

[CP.II.4] Among all the components building the NEMO model, two ocean tracers routines (advection[6] and diffusion[7]) are the most frequently executed. The advection kernel computes the current trend due to total advection of tracers using different schemes (e.g., the MUSCL or Monotone Upstream-centered Scheme for Conservation Laws), and adds it to the general tracer trend. The diffusion operator computes the horizontal tracer diffusive trend and adds it to the general trend of a tracer equation.

The number of passive tracers can in the order of dozens, especially when the biogeochemical component is active. In this regard, the shared memory parallelization of the advection and diffusion kernels could raise the overall performance. Although the MPI support has been improved (replacing point-to-point to collective primitives), the ocean tracers routines still represent a serious compute-specific bottleneck.

[CM.II.8] A prototype GPU implementation has been reported to achieve a 2.5x speedup factor when compared to a CPU-only implementation. However, in a MPI + multi-GPU environment performance does not scale and is similar to a MPI CPU-only implementation. This is mainly due to the needed CPU-GPU transfers when communicating GPUs among them, representing a communication bottleneck.

[ST.II.2] The CMCC-CM3 climate model will produce massive outputs of data that need to be stored for being processed by subsequent operations in the workflow. In addition, global data produced by the workflow needs to be accessed frequently and may also increase in size. Given the large size of the output (tens of gigabytes), this may create a potential storage bottleneck.

[6] http://forge.ipsl.jussieu.fr/nemo/browser/NEMO/trunk/src/OCE/TRA/traadv_mus.F90
[7] http://forge.ipsl.jussieu.fr/nemo/browser/NEMO/trunk/src/OCE/TRA/traldf_lap_blp.F90

Potential optimizations

Several optimizations can be applied to NEMO. At the function level, OpenMP can be used in order to exploit the internal parallelism of specific kernels, in particular, the two tracer functions identified. At a complementary level, a more advanced parallelization approach can be applied to the NEMO model in order to improve its performance. Indeed, the tile concept used in NEMO fosters parallelization, as tiles are totally independent of each other. Thus, MPI and OpenMP can be used to implement a hybrid-parallel solution, at the system level and the node level respectively.

A second potential improvement consists in migrating NEMO to run on GPUs. Indeed, there is an effort within the NEMO Community to leverage GPUs. This work relies on the adoption of a Domain Specific Language (DSL) for climate and weather modeling via PSyclone (developed by STFC) or DAWN (developed by MeteoSwiss, CSCS, ETHZ, and Vulcan). With this type of approach, most of the community can stay with the original code and not worry about new programming languages such as CUDA for GPUs. However, this path potentially sets a performance limitation since not all GPU features are guaranteed to be exploited by the DSL. Indeed, NEMO can be ported partially to GPU, specifically using CUDA, and adjusting the implementation to the underlying architecture of the target GPU. This is the case of the two identified tracers routines, which can be implemented and optimized on CUDA.

The limitation in the use of multi-GPU systems (CPU-GPU memory transfers) can be alleviated by the use of the NVIDIA Collective Communication Library (NCCL). This library is optimized for collective communications, as those used in NEMO, and allows for efficient direct access to GPU memory without the intervention of the CPU. Therefore, this solution can mitigate the problem identified in the current multi-GPU implementation. An alternative strategy consists in leveraging CUDA-aware MPI implementations, which can also avoid the use of CPU memory as a temporal buffer resource for communication between GPU devices.

For storage, new technologies such as NVRAM may help alleviate the bottlenecks for the output of the CMCC-CM3 climate model and for the frequent retrieval of global information of the Feature Extraction and Statistical Analysis workflow.

In this part of the workflow, the project will also aim to deploy ML-based schemes (e.g. based on NNs) for TC detection and tracking. This also entangles the necessity to consider NN-specific kernels (GeMM, Convolutions, activation functions) as possible compute-specific bottlenecks. The project will care about efficient implementations of these kernels on specific devices (GPUs, FPGAs and EPI).

5. Pillar III: Urgent Computing

Pillar III deals with Urgent Computing (UC) related problems. These are defined as problems that need HPC/HPDA (High Performance Data Analytics) systems immediately after an emergency situation, and typically combine complex edge-to-end workflows with capacity computing under strict time-to-solution constraints. Within the project, two UC applications are targeted: Tsunamis and Earthquakes. Both present the same sequential phases:

1. Pre-processing phase.
2. Simulation phase.
3. Post-processing phase.

In the pre-processing phase an ensemble of possible sources is defined based on seismic data assimilation and earthquake parameter estimation. In the simulation phase, individual cases are simulated for all the scenarios in the ensemble. Then, in the post-processing phase the simulation results are processed to produce probabilistic forecasts, and affectation maps. In both UC applications, the compute intensive parts appear in the simulation phase. We next describe the identified bottlenecks for each use case.

5.1 Tsunamis (PTF/FTRT)

The PTF/FTRT workflow is depicted in Figure III. The Tsunami use case relies on simulations performed using Tsunami-HySEA. This is a numerical model of the HySEA family specifically designed for quake-generated tsunami simulations. The model relies on GPUs to obtain a faster-than-real-time (FTRT) realisation. The model leverages CUDA (custom kernels) to implement an explicit numerical solver, involving small matrices, and does not rely on any linear algebra libraries. MPI is leveraged to enable multi-GPU simulations. Tsunami-HySEA is embarrassingly parallel and a stencil-type computation is performed on each GPU. Data locality within the GPU is exploited at the GPU block granularity.

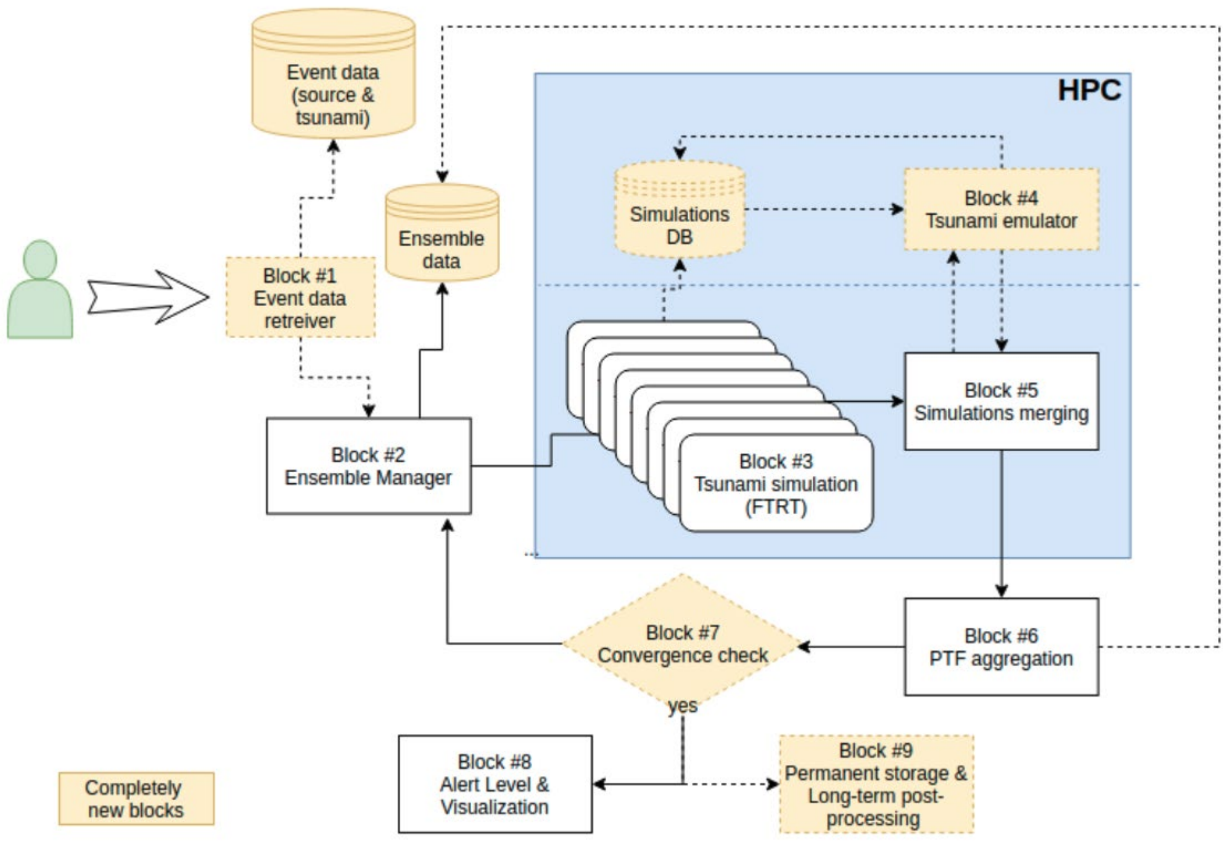
Event data (source & tsunami)
HPC
Simulations DB
Block #4 Tsunami emulator
Ensemble data
Block #1 Event data retreiver
Block #5 Simulations merging
Block #2 Ensemble Manager
Block #3 Tsunami simulation (FTRT)
Block #7 Convergence check
Block #6 PTF aggregation
yes
Block #8 Alert Level & Visualization
Block #9 Permanent storage & Long-term post-processing
Completely new blocks

Figure III. Schematic representation of the PTF/FTRT workflow. White boxes are building blocks already implemented. Yellow boxes are building blocks planned to be developed.

Two audits were recently performed within POP for the Tsunami-HySEA model [POP-HySEA1], [POP-HySEA2]. We next list the observations from the first audit:

- A synchronous MPI communication between processes was identified as the main limiting factor for scalability.
- Scalability issues, mainly due to load unbalanced, produce MPI waiting time.
- Point-to-point communications are limited by the network bandwidth.
- CUDA parallelization is fair but scaling problems appear because of the overhead resulting from launch kernels.

Therefore, the model was adapted to introduce asynchronous communications in order to overlap them with computation.

In the second audit, a GPU kernel was identified as a bottleneck. This kernel was improved and the bottleneck removed. The main observations from this second audit were:

- MPI point to point communications are overlapped with execution of kernels.
- MPI allows direct transfer to the devices (GPU).
- Communication bottlenecks appear due to inefficient GPU kernels.

With this second audit, specific kernels were identified and optimized, thus removing the remaining communication bottlenecks.

Identified bottlenecks

[CP.III.1] With the two reported analyses, the Tsunami-HySEA is a well balanced and optimized model running on multi-GPU systems. Therefore, there are no clear compute-specific bottlenecks that can be addressed within the model itself. However, the number of simulations to perform in the PTF/FTRT workflow will be significant (from hundreds to thousands) and may induce a compute-intensive bottleneck, depending on the availability of HPC resources.

[CM.III.1] A potential communication-specific bottleneck in the PTF/FTRT workflow may appear in the transmission of the results from Block #3 to Block #5. The data transfers can be expensive, generating a significant communication overhead. Therefore, this is a potential source of bottlenecks in the Tsunami simulation.

[ST.III.1] The output produced by the Tsunami-HySEA may also impose a storage bottleneck due to its size and the large number of simulations to perform.

Potential Optimizations

Machine Learning to reduce Simulation Workload

The set of simulations performed in Block #3 (Tsunami simulation) of the PTF/FTRT workflow (see Figure III) can produce large amounts of stream data (simulation results), characterized by high volume and high generation rate (i.e., high frequency of simulations). This may require that the simulation aggregation task performed by Block #5 (Simulations merging) leverages machine learning (ML) algorithms for analysis of simulation output adopting efficient and scalable solutions to satisfy the required short-time constraint imposed by the Tsunami use case.

In fact, the simulation results, produced by Block #3 in the form of data streams, can be analysed by a parallel machine learning algorithm to keep the pace with the rate of generated data streams. In particular, a frequent pattern algorithm could be used to analyse the simulation results and detect the frequent patterns hidden in the data very quickly, thus discovering the knowledge models in a very short time. Conversely, a *frequent pattern* is a set of items, subsequences, or substructures that occur frequently together (or strongly correlated) in the data set. This could provide several benefits and additional insights to the tasks that are executed in Blocks #4 and #6. For example, assuming that simulations are run by tuning several input settings, frequent itemsets in the simulation results may correspond to some patterns that are invariant with respect to the different simulations. Typically, we may have hundreds of simulations (about 400 or 500). Therefore, although the simulations exploit parallelism at some degree, the total execution time may be high. In consequence, the use of a machine learning strategy to reduce the number of simulations without losing accuracy may present an optimization opportunity.

Considering the specific case we are dealing with, Block #3 of the PTF/FTRT workflow produces simulation results in the form of data streams. We can integrate a *pattern mining module* (PM, Pattern Miner) in this architecture to discover *(i)* frequent items and *(ii)* frequent itemsets from such data streams. The first task, namely frequent-item discovery, is very popular both for its simplicity and because it is often used as a subroutine to discover the frequent itemsets: its goal is to find, in a sequence of items, those whose frequency exceeds a given threshold *min_sup*. The second task consists in the discovery of *frequent itemsets* defined as a set of distinct items appearing together (or concurrently appearing) in a number of transactions whose frequency is equal or higher than *min_sup*. This task can be severely time-consuming, since the number of candidates is combinatorial with the size. The usual technique is to first discover frequent items, and then build candidate itemsets incrementally, exploiting the *Apriori* property, which states that an itemset can be frequent only if all of its subsets are also frequent. This becomes a crucial issue in our proposed machine learning module because, as previously mentioned, Block #3 may involve hundreds of simulations, computed sequentially or in parallel, thus producing high data volumes (simulation results) at high generation rates.

Potential improvement (Communication-related)

To speed up the transfer of data that compose the output of Block #3 to Block #5 for data simulation merging, data transmission can be done in streaming, by exploiting in-memory storage and bypassing database reading/writing just during this phase. However, the output data from Block #3 are simulation results that can be also used by Block #4. For this reason they still need to be stored in the simulation database. Because of this, Block #3 or Block #5 write these data in the Simulations DB. For example, after it merges simulation data, Block #5 just could send its output to Block #6. In this way, secondary storage access will not affect the speed of this computation phase, because it is done in parallel with the operations in Block #6. By streaming data coming from the simulation execution, the flow of data generated by various simulations can be managed and collected, overlapping simulation computation and data communication. By using stream processing techniques, data streams can be processed, stored, analysed, and acted upon as if they were generated in real-time. This approach may introduce concurrency between the execution and communication steps avoiding the serialization of those operations and minimizing the interactions between operations of Block #3 and Block #5.

If the streaming approach is considered, the computation of Block #3 and Block #5 can be seen as a Map-Reduce step that exploits parallelism both in simulation execution (map phase) and in simulation results merging (reduce phase). Map-reduce provides a general partitioning and processing mechanism to distribute aggregation workload across different processors, thus it could be appropriate for the HPC platforms we are considering. According to this strategy, Block #3 can be implemented as a set of mappers, which execute simulations from the input data in parallel, whereas Block #5 can be implemented as a set of reducers, which merge simulation results. To be aligned with the project, this can be implemented with the DDS library, which implements a spark-like syntax on top of PyCOMPSs.

MonteCarlo to reduce the number of simulations

The number of simulations in the PTF/FTRT workflow can be reduced using a MonteCarlo method to steer the selection of a reduced number of the simulation scenarios. This may need to be embedded into an iterative refinement process, with several steps required for convergence.

5.2 UC Integrated Services for EarthQuakes (UCIS4EQ)

Figure IV illustrates the workflows scheme for the earthquake use case (UCIS4EQ). Most of the compute intensive part is performed in Block #4 (HPC simulations) running on HPC infrastructure. Full waveform modelling and inversion simulation is performed by the SALVUS suite. The remaining blocks in the UCIS4EQ workflow are light-weight compared to Block #4, and perform acquisition, preparation of data, and postprocessing. Therefore, they do not represent a potential compute bottleneck.

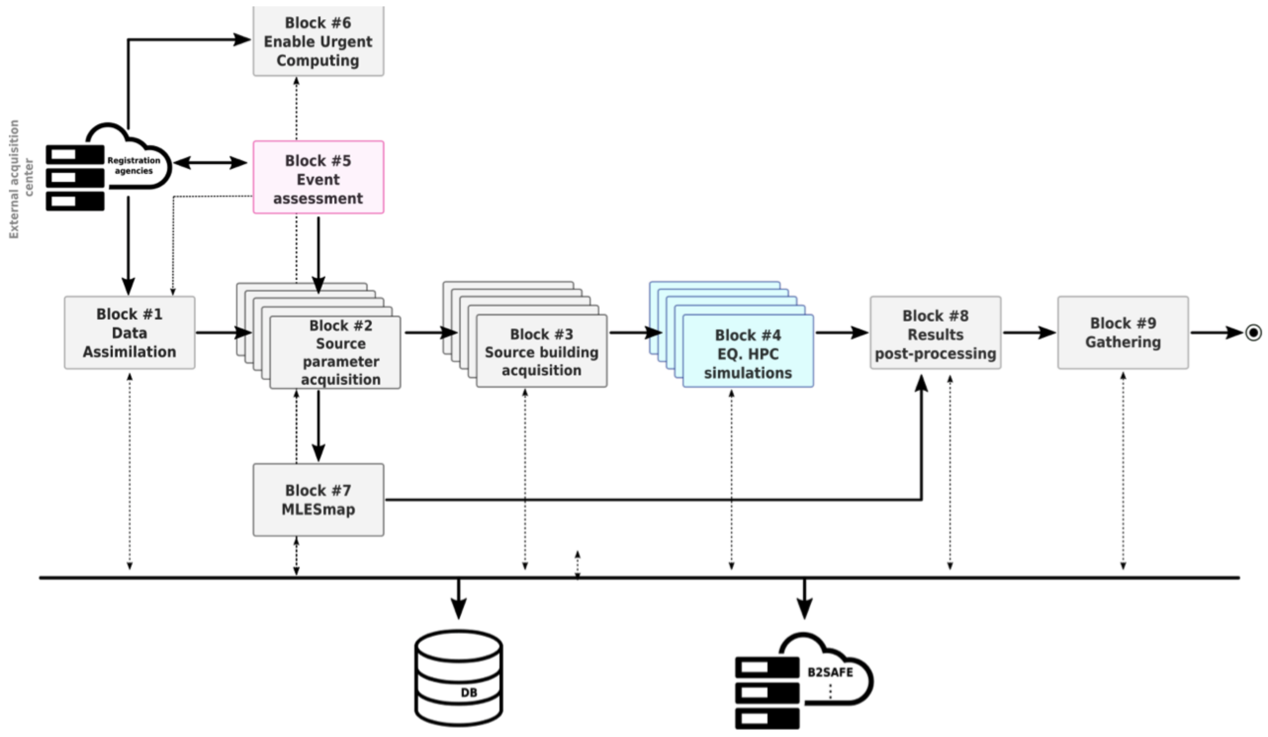

Figure IV. Schematic representation of UCIS4EQ workflow. Gray boxes correspond to building blocks already deployed in external servers or in the cloud. The blue box is deployed in an HPC facility. The pink box represents a building block needing data streaming deployment.

The SALVUS suite is proprietary software from Mondaic. The suite is optimized for HPC systems and can run either on CPU or GPU, using MPI as communication infrastructure between processes. SALVUS performance in GPUs has been profiled in POP [POP-SALVUS]. The following observations were provided in that report:

- Good weak scaling behaviour on the initial case set, with little idle times per GPU.

- No disadvantage experienced when using the bigger per GPU test case sizes.
- Three global synchronizations per iteration where computation overlaps reasonably well with the MPI communication.
- One of the compute-intensive kernels requires a "sufficiently large" workload to hide communication.

The following recommendations were provided:

- Check if the number of synchronizations per iteration can be reduced as they may cause communication overlap to fail at a higher scale.
- Run a production-size case and check whether the MPI process count becomes a noticeable problem.
- Compare performance of different meshes in terms of complexity and their influence on load balance.
- Run with I/O.

Given the observations in this report, and because of the proprietary nature of SALVUS, we do not identify potential compute-specific bottlenecks. We observe though that communication-specific bottlenecks may arise within the SALVUS suite for a production-size case.

Identified bottlenecks

[CP.III.2] Within UCIS4EQ workflow, Block #7 (MLESmap) will rely on NN models. This does not represent a compute-specific bottleneck as the NN training will be performed off-line. However, they may affect Block #4's previously identified compute-specific bottleneck since the output of Block #7 will be the input for the uncertainty quantification stage and will provide real-time affectation information prior to the availability of the simulated results obtained in Block #4. Therefore, we must regard Block #7 in general, and the use of NN in particular, as a potential attenuation factor for the Block #4 compute-specific bottlenecks in the UCIS4EQ workflow.

[CM.III.2] The UCIS4EQ workflow has two critical points where communication-specific bottlenecks may appear. They arise because several types of systems used in the workflow are not connected tightly. The first point is between blocks #3 and #4. Block #3 (source building) prepares input data for the HPC simulations. This block is not performed at HPC premises and, therefore, its output data needs to be sent to the HPC system where the simulations take place. The amount of data ranges between 1 GB and up to 15 GB in some cases. Thus, the transfer time of this data may represent a communication-specific bottleneck.

Similarly, the output data resulting from the HPC simulations at HPC premises need to be sent to non-HPC systems where data is post-processed in blocks #8 (Results postprocessing) and #9 (Gathering). In this case, the data can occupy tens of GB.

The remaining communication needs within the UCIS4EQ do not represent a bottleneck as they are performed within the same system and/or have much lower file sizes to transfer.

[ST.III.3] The UCIS4EQ workflow relies on HPC simulations (block #4), performed in the HPC facility, which are those that need the largest files to be used/produced (either as input or as output). The dimension of these output files may impose a storage-specific bottleneck.

[ST.III.4] In addition, UCIS4EQ relies on long term storage systems (DB and B2SAFE) where static and dynamic data are stored and retrieved. Dynamic data (data state along the workflow) does not represent a potential bottleneck for storage since its size is low. However, static data (history information, pre-trained NN models, output data produced by simulations) may represent a storage-specific bottleneck.

Potential Optimizations

A path to alleviate the communication-specific bottleneck around Block #4 (at its input and at its output) is to conduct block #3 (preprocessing) and blocks #8 and #9 (post-processing) in the HPC infrastructure. With this, the transfer of data will be performed within the HPC network and its higher bandwidth and lower latency will significantly reduce transfer time. Once this is achieved, other optimization strategies can be considered. One option is optimizing data partitioning and performing parallel data communication together with Map-Reduce strategies in the interaction between all these blocks. Additionally, the use of faster storage technologies (e.g. NVRAM) may reduce bottlenecks in the HPC system for the UCIS4EQ simulations.

6. All Pillars

In addition to the ML/NN specific compute-intensive kernels **[CP.X.1]** for CONV, GeMV, GeMM, depending on the type of NN that is more appropriate for each individual Pillar, we have identified two potential bottlenecks for communication and storage when dealing with NN training.

[CM.X.1] As we will rely on an HPC system with tens or hundreds of nodes, we will take advantage of a distributed training process. In this situation, it is of paramount importance to ensure the effectiveness of such distributed processes via an appropriate handling of synchronization and communication tasks. In this sense, an AllReduce communication is typically used in a distributed training process relying on data parallelism.

[ST.X.1] The second bottleneck stems from the fact that input data set for the training process needs to be partitioned and retrieved from disk into the computing nodes prior or during the training process. Proper access to the dataset will be also key for achieving an efficient training process.

7. Discussion

In this section we first compile all the identified bottlenecks from the workflows to then discuss the suitability of using heterogeneous architectures, communication technologies or storage technologies to tackle such bottlenecks.

Figure V displays a summary of the identified bottlenecks. Compute-related bottlenecks are shown there in green, communication-related bottlenecks in blue, and storage-related bottlenecks in orange. The bottom part of the figure shows all the bottlenecks identified in the different simulation frameworks and models (KRATOS, Randomized SVD, FESOM2, OpenIFS, NEMO, Tsunami-HySEA, and SALVUS). No clear bottlenecks were identified for Tsunami-HySEA because the application is fully optimized and embarrassingly parallel. The same issue applies to SALVUS, in this case also because this is part of proprietary software and we have no access to it. On top of each framework we show, for each pillar, bottlenecks identified at workflow level.

The fact that all workflows rely on hundreds of compute-intensive simulations for the input generation phase is of particular interest. This represents an orthogonal compute-intensive bottleneck for all workflows. Indeed, the three pillar workflows plan to address these overheads via the use of NNs and machine learning (ML) algorithms for effective prediction and pruning of ensemble members, though following a different approach in each case. In the top part of the figure, we highlight this fact by showing some of the most demanded types of operations in ML/NN. Specific kernels appearing in the pillars will need to be implemented efficiently to prevent bottlenecks.

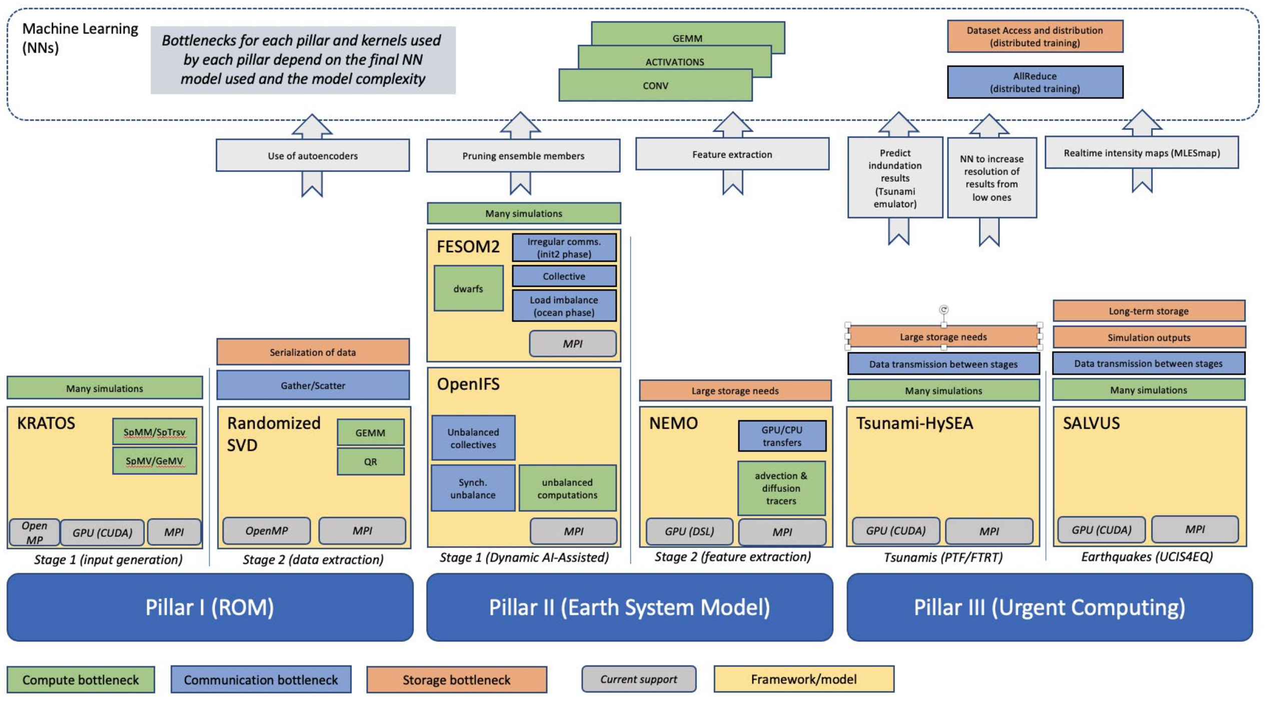

Figure V. Bottlenecks identified for the three Pillars as a result of the work performed in task T3.1. The bottlenecks are classified as compute-intensive, communication and storage using different colors: green, blue and orange, respectively. The figure also specifies the current support for each framework/model.

7.1 GPU support

Figure V also lists the current architectures currently supported by the simulation frameworks/models. For instance, neither FESOM2 nor OpenIFS use GPUs in their computations. This offers an opportunity to alleviate compute-intensive bottlenecks.

Similarly, although NEMO has some side implementations with GPUs (using a DSL), the target version within the project has no GPU support. Thus, the workflow for Pillar II may benefit from an extensive use of GPU within their frameworks. It will be an interesting exercise to dig deeper into the two identified kernels in NEMO and assess how they behave when ported to GPU using CUDA.

The two frameworks/models for Pillar III (Tsunami-HySEA and SALVUS) use GPUs for the computations. Indeed, in both cases GPUs are highly utilized in an efficient way and no bottlenecks were reported.

Finally, we note that GPUs can be used for the development of NN models for the three pillar workflows. GeMM and CONV operations can run two to three orders of magnitude faster than the same operation executed on CPUs. Therefore, it will be vital for the project to leverage GPUs for the training process of NN models for the pillar workflows.

7.2 FPGA support

From Figure V we can appreciate that none of the workflows use FPGAs in any stage. This is a clear sign that FPGAs are not yet considered a first-class citizen in HPC. However, in some situations, the use of FPGAs may provide benefits for the workflows, probably not in terms of raw performance but, being a highly energy efficient architecture, in performance-per-Watt.

As a first possibility, the workflow for Pillar I relies on the AMGCL library within KRATOS for sparse matrix-matrix and matrix-vector multiplications as well as sparse triangular solve (SpMM, SpMV, SpTrsv). FPGAs are rather efficient when the problem to solve is irregular, and sparsity indeed introduces some type of irregularity in memory access patterns. FPGAs will be explored within Pillar I for the efficient implementation of the SpMM, SpMV, SpTrsv kernels. This will be combined with the use of mixed precision arithmetic, which can be easily customized and adapted in an FPGA.

Alternatively to the use of FPGAs in Pillar I, the other two pillars may benefit from the use of FPGAs when implementing their NN models. Indeed, in both cases, a trained NN model will be inferred in order to prune running simulations. This inference process can be performed in an FPGA in a considerably more energy-efficient manner. Within this project we will tackle the use of FPGAs for the inference with NNs applied to the pillar workflows.

7.3 EPI support

The availability of commercial RISC-V instruction set architecture (ISA)-based processors with complete functionality is very limited or even nonexistent at the moment. This is especially the case for processors equipped with SIMD floating point units (FPUs) or hardware vector accelerators. It is difficult to foresee how fast this situation will change during the development of the project. For this reason, we plan to rely mostly on software simulators and FPGA-enabled designs in order to explore the performance and energy efficiency of the RISC-V ISA-based processors.

We consider compute-intensive kernels to be especially interesting candidates to investigate on the RISC-V designs. These include, for example, GeMM and SVD for Pillar I. In addition, NN-related kernels will be ported to EPI in order to provide support for Pillar II and Pillar III developments.

7.4 Final List of Bottlenecks

Table II summarizes the identified bottlenecks in each of the project Pillars. The Table also shows the type of bottlenecks and the potential optimization that can be applied within the project to alleviate them. Applicability of different heterogeneous architectures (GPU, FPGA, EPI) are indicated at the bottleneck level. Also, the complexity degree (Low, Medium, High) and potential benefits (Low, Medium, High) are specified for each bottleneck.

Table II. Final summary of bottlenecks identified for the three Pillars.

Pillar/ Use Case	Bottleneck Type	Bottleneck/Optimization	Applicability					Complexity (L/M/H)	Benefit (L/M/H)
			GPU	FPGA	EPI	Storage	Network		
Pillar I ROM	CP.I.1	AMG preconditioner / SpMV / SpTrsv GMRES / GeMV GPU: Accelerate memory-bound operations via GPU FPGA: Use customized precision	Yes	Yes				High	High
	CP.I.2	Hundreds/Thousand of simulations Train a Neural Network/Machine Learning model and perform simulation pruning	Yes	Yes				High	High
	CP.I.3	Randomized SVD: GeMM / QR GPU: Accelerate computation EPI: Use wide SIMD (vector) instructions	Yes		Yes			Medium	Low
	CM.I.1	Gathering/Scattering data from multiple computing nodes in randomized SVD Improve transition from distributed to centralized format in ds-array					Yes	Medium	Low
	ST.I.1	Serialization of data in PyCOMPSs Improve random generation of ds-array				Yes		Medium	High
Pillar II OpenIFS	CP.II.1	Unbalanced computations							
	CM.II.1	Unbalanced Collective Communications Introduce OpenMP-based parallelization					Yes	High	High
	CM.II.2	Process synchronization Introduce OpenMP-based parallelization					Yes	High	High
	CM.II.3	OpenIFS-Nemo Will not be used in the project					Yes	NA	NA
	CM.II.4	OpenIFS-Nemo Will not be used in the project					Yes	NA	NA
Pillar II FESOM 2	CP.II.2	FESOM2 dwarfs Porting to GPU and FPGA	Yes	Yes	Yes			High	Medium
	CM.II.5	init2 irregular communications Avoid initialization algorithm to avoid irregular pattern with produces load imbalance in communications					Yes	Medium	Medium
	CM.II.6	Large collective operation not scaling Reduce MPI operations					Yes	Medium	Medium
	CM.II.7	Load imbalance in ocean phase Review the domain decomposition algorithm to improve load balance					Yes	Medium	Medium
Pillar II OpenIFS & FESOM 2	CP.II.3	Hundreds/Thousand of simulations Train a Neural Network/Machine Learning model and perform simulation pruning	Yes	Yes				High	High

Pillar II NEMO	CP.II.4	Advection and diffusion tracers Porting to GPU	Yes					Medium	High
	CM.II.8	Multi-GPU communication using the CPU memory Using NCCL as communication protocol between GPUs	Yes					High	Medium
	ST.II.1	Large storage needs Use NVRAM technology				Yes		Medium	Medium
Pillar III Tsunamis	CP.III.1	Hundreds/Thousand of simulations Train a Neural Network/Machine Learning model and perform simulation pruning Use MonteCarlo to reduce number of simulations	Yes	Yes				Medium	High
	CM.III.1	Data transmission between stages Streaming transmission					Yes	Medium	Medium
	ST.III.1	Large storage needs Use NVRAM technology				Yes		Medium	Medium
Pillar III Earthquakes	CP.III.2	Hundreds/Thousand of simulation runs MLESmap (NN model) to perform simulation pruning	Yes	Yes				Medium	High
	CM.III.2	Data transmission between stages Run preprocessing and postprocessing stages in HPC system (on premise)					Yes	Medium	Medium
	ST.III.3	Large output of simulations Use NVRAM technology				Yes		Medium	Medium
	ST.III.4	Large dimensions of static data Use NVRAM technology				Yes		Medium	Medium
Neural Networks (all Pillars)	CP.X.1	NN kernels Optimize implementation of the kernel for both training and inference FPGA: Energy Efficient implementation	Yes	Yes	Yes			Medium	Medium
	CM.X.1	Distributed training inefficient due to communication issues Efficient implementation of data parallelism Distributed training with specialized libraries (NCCL)					Yes	Medium	High
	ST.X.1	Dataset access in distributed training process New storage technology (NVRAM) to reduce access latency				Yes		Medium	Medium

8. Conclusion

In this document we have addressed the identification process we performed in Task 3.1 for the potential bottlenecks that exist or may exist for the use cases defined by the three Pillars. Most of the work consisted in meetings with the Pillar's partners and collection of a set of previous reports where specific components of the Pillar's use cases were already analysed.

The three Pillars are rather different but they end up showing related bottlenecks for the three axes we analysed (compute, communication, storage). The identified bottlenecks have been listed and potential optimizations and strategies for addressing them have been identified for each bottleneck. Also important, we have matched them with new technologies that will be used within the project.

The final Table of bottlenecks and expected actions will steer the subsequent development-related tasks within WP3.

9. Acronyms and Abbreviations

-	AI	Artificial Intelligence
-	AMG	Algebraic Multigrid
-	CESM	Community Earth System Model
-	CMIP6	Coupled Model Intercomparison Project Phase 6
-	CPU	Central Processing Unit
-	CONV	Convolution
-	CUDA	Compute Unified Device Architecture
-	D	Deliverable
-	DA	Data Analytics
-	DSL	Domain-Specific Language
-	EPI	European Processor Initiative
-	ESM	Earth System Model
-	ETHZ	Eidgenössische Technische Hochschule Zürich
-	FEM	Finite Elements Method
-	FESOM2	Finite Element Sea Ice-Ocean Model 2
-	FPGA	Field Programmable Gate Array
-	FPU	Floating point unit
-	FTRT	Faster Than Real Time
-	GB	Giga Byte
-	GeMM	General Matrix-Matrix product
-	GeMV	General Matrix-Vector product
-	GPU	Graphics Processing Unit
-	HPC	High Performance Computing
-	HPDA	High Performance Data Analytics
-	HySEA	Hyperbolic Systems and Efficient Algorithms
-	I/O	Input/Output
-	ML	Machine Learning
-	MLESmap	Machine-Learning based Estimator for ground motion Shaking maps
-	MPI	Message Passing Interface
-	MUSCL	Monotone Upstream-centered Scheme for Conservation Laws
-	NA	Not applicable
-	NCCL	Nvidia Collective Communication Library
-	NEMO	Nucleus for European Modelling of the Ocean
-	NN	Neural Network
-	NVRAM	Non-Volatile Random Access Memory
-	OpenIFS	Open Integrated Forecasting System
-	PAPI	Performance Application Programming Interface
-	POP	Performance Optimization and Productivity
-	PRACE	Partnership for Advanced Computing in Europe
-	PTF	Probabilistic Tsunami Forecast
-	ROM	Reduced Order Model
-	SALVUS	Spectral-Element Wave Propagation (software package)
-	SOR	Successive Over-Relaxation
-	SpMV	Sparse Matrix-Vector product
-	SpMM	Sparse Matrix-Matrix product
-	SpTrsv	Sparse Triangular Solve

- STFC	Science and Technology Facilities Council
- SVD	Single Value Decomposition
- TC	Tropical Cyclone
- UC	Urgent Computing
- UCIS4EQ	Urgent Computing Integrated Services for Earthquakes
- XIOS	XML-IO-Server
- WP	Work Package

10. References

[OPENIFS-BSC] Mario. C. Acosta, “Profiling and Computational Performance of IFS using BSC Tools”

[REP-FESOM2] Mario C. Acosta, FESOM2 Finite volumE Sea ice Ocean Model enhancement, Expert Report PRACE Preparatory Access Type C & D, Project #2010PA5513, Type C, June 2021

[Koldunov19] N. V. Koldunov, V. Aizinger, N. Rakowsky, P. Scholz, D. Sidorenko, S. Danilov, and T. Jung, Scalability and some optimization of the Finite-volumE Sea ice–Ocean Model, Version 2.0 (FESOM2), Geosci. Model Dev., 12, 3991–4012, http://doi.org/10.5194/gmd-12-3991-2019, 2019.

[ISENES3-NEMO] IS-ENES3 Milestone M8.4, Definition of NEMO optimization strategy, January 25 2021.

[POP-NEMO] NEMO (POP_AR_078) report, April, 11 2020

[PRACE-NEMO] S. V. Paranuzzi, M.C. Acosta, M. Caastrillo, O. Tintó, K. Serradell, Keeping computational performance analysis simple: an evaluation of the NEMO BENCH test, April 20, 2020.

[Mar20] P. G. Martinsson and J. Tropp, Randomized Numerical Linear Algebra: Foundations & Algorithms. Acta Numerica, **29**, pp 403-572, 2020.