H2020-INFRAEDI-2018-2020

The European Centre of Excellence for Engineering Applications

Project Number: 823691

D4.6

Report on Enhanced Services Progress

The EXCELLERAT project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823691

Workpackage:	4	Enhanced services
Author(s):	Niclas Jansson	KTH
	Gavin Pringle	UEDIN
	Dennis Grieger	USTUT
	Christian Gscheidle	Fraunhofer
	Janik Schüssler	SSC
Approved by	Executive Centre Management	
Reviewer	Claudio Arlandini	CINECA
Reviewer	Maike Gilliot	TERATEC
Dissemination Level	PU	

Date	Author	Comments	Version	Status
2020-10-27	N. Jansson et al.	Initial contributions	V0.1	Draft
2020-10-31	N. Jansson et al.	Draft ready for QA	V0.2	Draft
2020-11-20	N. Jansson et al.	Updates from first QA round	V0.3	Draft
2020-11-27	N. Jansson et al.	Updates from second QA round	V0.4	Draft

List of abbreviations

3D	*Three-dimensional*
AMR	*Adaptive Mesh Refinement*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
AVX	*Advanced Vector Extensions*
CoE	*Centre of Excellence*
CFD	*Computational Fluid Dynamics*
CRM	*Common Research Model*
CUDA	*Compute Unified Device Architecture*
DMD	*Dynamic Mode Decomposition*
EPI	*European Processor Initiative*
FPGA	*Field Programmable Gate Array*
FPU	*Floating Point Unit*
GLL	*Gauss Lobatto Legendre*
GPR	*Gaussian Process Regression*
HDF5	*Hierarchical Data Format 5*
HIP	*Heterogeneous-compute Interface for Portability*
HPC	*High-Performance Computing*
ISA	*Instruction Set Architecture*
I/O	*Input/Output*
JOSS	*Journal of Open-Source Software*
MKL	*Math Kernel Library*
ML	*Machine Learning*
MPI	*Message Passing Interface*
MPMD	*Multiple-Program, Multiple-Data*
NetCDF	*Network Common Data Form*
OpenACC	*Open Accelerators*
OpenMP	*Open Multi-Processing*
PAPI	*Performance Application Programming Interface*
PCA	*Principal Component Analysis*
PCE	*Polynomial Chaos Expansion*
PETSc	*Portable, Extensible Toolkit for Scientific Computation*
POD	*Proper Orthogonal Decomposition*
PVC	*Precessing Vortex Core*
QoI	*Quantity of Interest*
RANS	*Reynolds-averaged Navier-Stokes equations*
SENSEI	*Scalable in situ analysis and visualization*
Sr	*Strouhal Number*
SHM	*Shared Memory*
SPMD	*Single-Program, Multiple-Data*
SVD	*Singular Value Decomposition*
TCP	*Transmission Control Protocol*
TGCC	*Très Grand Centre de calcul du CEA*

UI	*User Interface*
UQ	*Uncertainty Quantification*
V&V	*Validation & Verification*
VTK	*The Visualization Toolkit*
XDMF	*eXtensible Data Model and Format*

Executive Summary

This deliverable reports on the progress and significant achievements in work package 4 of the EXCELLERAT project. The work package deals with the development of EXCELLERAT's enhanced services; co-design, visualization, data analytics and management, in essence developing tools for an application's entire lifecycle.

The significant achievement in this reporting period has been to formulate prototypes of the enhanced services outlined in deliverable D4.2; scalable in-situ visualization workflows for both interactive and non-interactive analysis, in-situ data analytics and uncertainty quantification frameworks and a newly designed platform for HPC specific transfer and data management. These services have been developed and implemented based on use-case’s needs, either using derived model problems or scaled-down formulations of the full use-cases.

The plans for the remainder of the project are to continue developing the enhanced services, advanced past, current prototypes and focus on the integration of the development into the full use-cases in EXCELLERAT.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This document reports on the progress and significant achievements in work package 4 of the EXCELLERAT project. The work package deals with EXCELLERAT's enhanced services; co-design, visualization, data analytics and management, in essence developing tools for an application's entire lifecycle.

As described in deliverable D4.2 *Report on the Service Portfolio,* tools and methods developed within this work package are derived based on the common needs of EXCELLERAT's use-cases for solving engineering simulations at scale, as outlined in their so-called user-stories. Given the diversity of EXCELLERAT's core-codes, particular focus is laid on formulating services which are both code- and application-agnostic, enabling reuse and integration into several of the core-codes. Marketable services developed in this work package will be identified by work package 1 for inclusion in EXCELLERAT's service portfolio.

The deliverable is structured as a progress report. Each task in work package 4 give details of the work done, and progress towards realising the envisioned services outlined in deliverable D4.2 and outline future work for the remainder of the project.

2 Task 4.1: Co-design

EXCELLERAT follows an indirect co-design paradigm, wherein Core Partners gain access to the early release of state-of-the-art hardware, where this hardware is available typically due to a close working relationship between the vendor and the Core Partner. This avoids a *direct* co-design paradigm, as vendors typically will not alter their hardware to benefit a small set of applications. This indirect co-design paradigm permits EXCELLERAT to exploit trends in software and hardware and match them to the code design issues of our Reference Applications running our use cases.

The co-design working group agreed a methodology:

- Firstly, target Reference Applications are chosen.
- Full applications may be reduced to mini-apps,
    - where each mini-app is a small bundle of highly portable source code, with example input and output files,
    - which retains the computational characteristics of the full simulation, e.g., computational kernels are retained
        - perhaps key data movements epochs, such as I/O, are retained.
    - This can circumvent irrelevant issues that can arise when porting to novel architectures.
- The full code, or mini-app, is then ported to existing hardware or emulators,
    - associated libraries may also need to be installed
- Target simulation is also provided
    - ported code is optimised if time permits
- Initial profiling is then performed to locate kernels of interest,
- Profiling and performance are measured
    - using emerging libraries where possible,

- Code adaptations and/or improvements are investigated and reported back to owners.

We have created and managed a Co-Design Working Group, which monitors all co-design activities. These activities occur across many other Tasks, specifically,

- T3.1: Node-level performance optimisation,
- T3.2: System-level performance towards exascale,
- T3.4: Test lab for emerging technologies,
- T3.5: Validation and benchmarking suites,
- T4.1: Co-design,
- T4.3: Data analytics,
- T5.5: HPC service provisioning,

where T4.3 provides key linear algebra routines to test alongside our Reference Applications. T3.4 helps locate and document emerging technologies, T5.5 provides access to cutting edge platforms or emulators of future platforms, and T3.5 helps to determine bottleneck kernels via profiling the reference applications. T3.1 and T3.2 perform the node-level/accelerator tests. Lastly, it should be noted that we outsource required effort through our ongoing collaborations with the POP CoE [1].

During the last year, the target Reference Application codes involved in co-design have been identified as Alya, AVBP, CODA, Nekbone (Nek5000), and a particular dense linear algebra SVD solver from ScaLAPACK, namely *pdgesvd*, as this is the key routine employed by the DMD method, described in Task 4.3 Data Analytics (see Section 4.3).

Our living document started life as a Word document in our BSCW, but this enforced a debilitating single-editor bottleneck, thus the document was moved to our Wiki, which permits multiple concurrent editors.

As planned, as part of our living document, we have a Section containing a crib-sheet for authors to prepare their codes for exascale, for portable optimisations that are not tied to any particular hardware. This was produced, in collaboration with another CoE, namely CompBioMed [2], and is included in both the live working group document on our Wiki and in the Appendix of this deliverable (see Section 9.1).

Finally, we had planned to organise a joint Birds of a feather (BoF) session with ETP4HPC [3] at ISC'20; however, this was postponed due to Covid-19. It is planned to hold this BoF session at ISC'21 or a virtual workshop in 2021.

The remainder of this Section describes the novel hardware currently employed by the co-design working group, along with a high-level overview of the status of each of the participating Reference Applications.

2.1.1 Cutting edge hardware employed to date

A full list of current hardware available to the CoE and is maintained by Task5.5 is available in the wiki. Regarding co-design, the hardware involved to date are as follows:

- AMD
  - AMD EPYC cluster at TGCC (Très Grand Centre de calcul du CEA) [4].
  - AMD CPU and GPU clusters at DLR
- ARM
  - ARM Cavium ThunderX2 cluster from HPE at EPCC [5].
  - ARM Marvell ThunderX2 and Nvidia Tesla GPUs, ARMIDA, from EPI at E4 [6].
  - ARM thunderx2 cluster at TGCC (Très Grand Centre de calcul du CEA) [4].
- HPE SGI
  - GPU cluster, JEANZAY system at IDRIS [7].
- Huawei
  - Huawei "hi 6474 early silicon" cluster, JUAWEI, from EPI at Jülich Supercomputing Centre.
- IBM
  - Power9 plus NVIDIA Volta GPUs, Marconi100, at CINECA [8].
  - Power9 + GPUs Cluster, CTE, at BSC [9].
- Intel
  - Intel CPU cluster, Galileo, at CINECA [10].
  - Lenovo NeXtScale platform, featuring Intel Xeon Skylake (SKL) processors, Marconi A3 partition, at CINECA [11].
- NEC
  - SX-Aurora Vector Machine at HLRS [12].
- Xilinx
  - Xilinx's Alveo U280 FPGA cluster at EPCC [13].

2.2 Alya/BSC

In this second year we have focused in comparing the OpenACC and CUDA implementations to measure the additional performance that is available by using a lower level implementation strategy. These experiments have been carried out on tailored mini-apps generated to reproduce the assembly and linear solver phases of the time-integration.

Despite the better performance obtained with the low-level optimal kernels, those are not our preferred option because they complicate the maintenance and further development of the code, especially on the parts of the code related with discretization and modelling methods, such as turbulence models, which are maintained by application scientists rather than computer scientist. Nonetheless, this study provides us a sort of ideal performance reference for the GPU version of the code.

POWER 9 + NVIDIA V100 cluster

We tested our new min-app on the POWER9 CTE cluster of the Barcelona Supercomputing Center. This platform has an architecture similar to the Summit supercomputer, but with 4 GPUs instead of 6 GPUs per node. For the mini-app related with the assembly phase, the benefits of the low-level implementation were not only caused by a better tuning of the implementation but also an element-specific implementation. As a consequence, speedups of up to 2x were obtained. For the algebraic solver, the element-specific factor was not present, so the low-level implementation outperforms the directives-based approach only by 15%.

For more information on this work, and others outlined below, please see D3.2 Report on Exa-Enabling Enhancements and Benchmarks, and the Section on T3.4 Test Lab for Emerging Technologies.

Plans for the next 12 months

- Test Alya on new pre-exascale EuroHPC architectures
- Test Alya on novel ARM CPUs

Alya on FPGAs

We are currently working on porting Alya to FPGAs and as an initial step have performed some focussed experimentation. This has enabled us not only to identify the most time intensive kernel of the code, but furthermore to understand whether it is likely to be of benefit on the FPGA or not.

We found that the *nsi_element_operations_fast* kernel accounts for around 78% of the runtime with one of the standard test-cases which models a sphere and contains 16 million elements. This kernel is called for every single element, and constructs the matrix of equations that will then be solved. Whilst the individual executions of the functions are quick, the fact that it is executed so frequently accounts for such a significant runtime (around 90 seconds for every element which is performed twice per timestep). We profiled this using the PAPI library and a tool to interpret these results that has been developed as part of EXCELLERAT, which performs analysis on the performance counters and generates a focussed report of metrics that are important to HPC codes, and the summary is illustrated in Figure 1.

This profile is for the specific, *nsi_element_operations_fast*, subroutine and it is telling us that on average the routine is providing 1.54 GFLOP/s performance, which is very low, with the CPU Floating Point Units (FPUs) at less than 50% occupancy. Furthermore, the CPU is stalling for over 10% of the time (doing no work) and most of this (7.6% of all cycles) are due to memory stalls where the CPU has to wait for data to be made available. Furthermore, there is an imbalance of instructions, where for every floating-point calculation there are approximately two data reads. Whilst it is still fairly early on this work, we believe that this summary means that there is potential here for FPGA acceleration, whereby reworking the kernel as a dataflow algorithm we can significantly improve the FPU occupancy and reduce the number of overall stalls. We can also significantly increase the amount of floating-point capability, and it can be seen that currently the CPU is not vectorising these properly (only one single floating-point operation per FPU per cycle, rather than the 8 that it is theoretically capable of.) There are a number of dependency issues within the algorithm that need to be addressed to port onto the

FPGA.- This is our current area of focus, and from this early study we think it is fairly likely that the technology will show some benefit.

```
Total number of calls: 30 (30 profiled) and 1 dynamic memory allocations
Number of counter updates: 30 (30 possible, 100.0%)
Total execution time 2108.94 seconds, total number of billion cycles 7767.21
---------------------------------------
89.08% of cycles issued uop for execution, 77.58% cycles retired a uop

Average MIPs of 11147.56 and 0.0 single MFLOP/s and 1540.12 double MFLOP/s
   Total number of single precision MFLOP: 0.0 (N/A% 256B, N/A% 128B, N/A% scalar)
   Total number of double precision MFLOP: 3248029.72 (0.0% 256B, 0.0% 128B, 100.0% scalar)
   FPU occupancy: 41.82%

Total data read from DRAM 230829.58 MB, with 80480.12 MB on L3 miss
   Memory load operations that hit in L1: 99.61% and 0.19% missed
   Memory load operations that hit in L2: 0.15% and 0.04% missed
   Memory load operations that hit in L3: 0.02% and 0.02% missed
   Memory load operations that hit in FB: 0.22% and 0.02% went to DRAM
   Prefetcher performance:
      Hit in L2 cache 66.57%
      Missed in L2 cache 33.43%
      Evicted without being used (useless) 1.25%
   Ratio between FP and read instructions: 0.55
   Ratio between load and store instructions: 2.23

Cycles stalled 10.9% and 7.6% with memory access outstanding
   Cycles stalled with L1D miss 4.13%
   Cycles stalled with L2 miss 4.05%
   Cycles stalled with L3 miss 3.43%
```

Figure 1: Profile of Alya running on a FPGA.

2.3 AVBP/CERFACS

The original ABVP mini-app was based on the initial use cases of this CoE. However, to account for the dynamic mesh adaptation structure, since the beginning of project 80% of the source code of ABVP was rewritten. Hence, the original mini-app no longer reflects the main code and can no longer be used.

Now that AVBP 7.7 has been released with GPU support, we will re-evaluate this position and see if we can extract a new mini-app. However, to date, a mini-app has not been required due to AVBP's ease of portability.

ARM cluster

AVBP has been tested on the INTI cluster at TGCC, FULHAME cluster at EPCC and JUAWEI cluster at JSC from EPI (hi 6464 Huawei early silicon). Performance is on track with current user experience on x86 clusters but is highly dependent on the compiler vendor and compiler version. Partial results can be found here [14] and consolidated results will be published in 2021.

AMD cluster

AVBP has been ported and tested up to 132k cores in the PRACE system IRENE from TGCC equipped with EPYC 2 AMD processors with near perfect scaling. See Figure 2, which shows strong scaling of AVBP on IRENE, for both an explosion simulation (circles) and a turbulent channel simulation (triangles) versus ideal acceleration. Access to the HLRS cluster (Hawk) has not been granted yet as the machine was delayed.

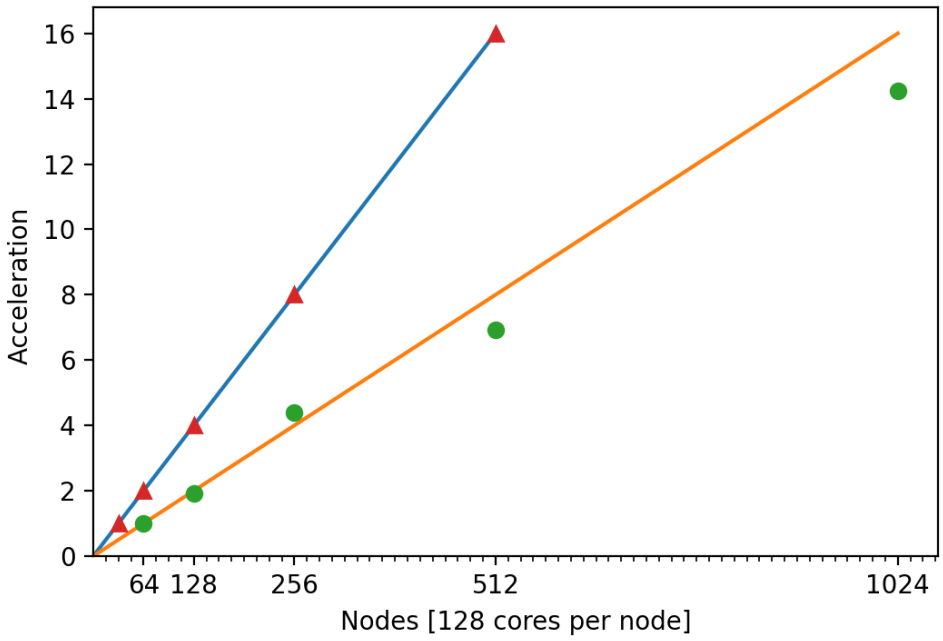


Figure 2: AVBP strong scaling on IRENE, for explosion and a turbulent simulations.

Power9+GPU clusters

Access to the CTE IBM Power9 system from BSC equipped with V100 GPUs is still pending.

Nvidia GPU cluster

CERFACS has been granted an early access to the new GPU partition on the JEANZAY system (Tier 1 - IDRIS) in France with 300k CPU hours to optimise and perform the aeronautical combustion chamber use Case. We are working jointly with IDRIS, HPE and Nvidia on the optimisation of the code for this activity.

For the plans for the next 12 months

- Optimisation of AVBP on V100 and test on A100 GPUs
- Test AVBP on RISC-V in collaboration with EPI.
- Test AVBP on Fujitsu A64FX ARM based processors
- Test AVBP on AWS Graviton ARM based processors

2.4 CODA/DLR

To test CODA's computational kernel on novel architectures and accelerators a mini-app has been created, thereby removing the need to adapt the entire workflow of CODA. This mini-app contains a set of representative benchmarks that evaluate the Sparse Linear Systems Solver library (Spliss), which runs CODA's computationally intensive linear solver.

GPU cluster

The work carried out focused on the porting of Spliss to GPUs, which has been achieved. After DLR's GPU cluster was put into operation during this period, first performance results of Spliss on GPUs were evaluated and up to 25x runtime improvement for initial benchmarks was achieved. Currently, Spliss is extended to support the efficient usage of multiple GPUs per compute node. Spliss is now ready to be used by CODA, whereas the computation in the linear solver can be transparently switched between CPU and GPU. Upcoming work will focus on testing and evaluating the entire workflow of CODA with the linear solver running on GPUs.

AMD cluster

After DLR's new AMD HPC cluster CARA went operational in February 2020, CODA and the surrounding workflow were installed and intensively tested. After identifying the ideal hybrid setup and adapting all workflow components to CARA, efforts were focused on evaluating the scalability of CODA on CARA using the EXCELLERAT use cases C6U1 and C6U2. The use case solves the Reynolds-averaged Navier-Stokes equations (RANS) with a Spalart-Allmaras turbulence model in its negative form (SA-neg). It uses finite volume spatial discretization with an implicit Euler time integration. The input of the use case is an unstructured prism mesh from the NASA Common Research Model (CRM) with about 5 million points and 10 million volume elements. The mesh is a rather small mesh chosen for strong scalability analysis of CODA at reasonable scales. Production meshes are at least 20 times larger and accordingly achieve a good efficiency on much higher scales. For the small mesh, the use case achieves about 60% parallel efficiency on the largest available partition on CARA with 512 nodes and 32,768 cores.

After getting access to DLR's new AMD HPC system CARA based on AMD's EPYC architecture and an Intel Cascade Lake test system during the period, both systems were evaluated with the use case. The initial results were compared, and we identified the ideal hybrid MPI-OpenMP setup for both architectures. Furthermore, we found a limitation in the AMD EPYC architecture that limits the efficient hybrid usage to four OpenMP threads per MPI

one. There is currently no plan to publish more on this result. This restricts CODA's hybrid capabilities and, thus, also its scalability since CODA relies on using as many OpenMP threads per MPI rank as possible. The Intel Cascade Lake architecture did not impose those limitations.

2.5 Nekbone (Nek5000)/CINECA,KTH,UEDIN

The Reference Application Nek5000 has a mini-app entitled Nekbone [15] which is used for co-design.

2.5.1 FPGA at EPCC

Work was undertaken exploring the role of FPGAs to accelerate Nekbone, both in terms of performance and also power efficiency. This was undertaken using Xilinx's latest Alveo U280 FPGA, and comparisons were made against a 24-core Intel Xeon Platinum Cascade Lake CPU and NVIDIA V100 GPU. Details around the optimisations' steps are provided in D3.2 *Report on Exa-enabling enhancements and benchmarks* and explored in depth in [16].

The table below (Table 1) contains performance and energy efficiency comparison of multiple kernels against other technologies, and illustrates a summary of results achieved. Running over all 24 CPU cores resulted in an energy efficiency of 0.37 GFLOPS/Watt. For comparison, we also include a single core CPU run, which resulted in 5.38 GFLOPS and energy efficiency of 0.08 GFLOPS/Watt. GPU performance was 407 GFLOPS and, due to the high performance, an energy efficiency of 2.34 GFLOPS/Watt. The GPU's performance is impressive, although it should be noted that the bespoke GPU acceleration in Nekbone has been developed and tuned over many years and GPU generations.

Description	Performance (GFLOPS)	Power usage (Watts)	Power Efficiency (GFLOPS/Watt)
1 core of CPU	5.38	65.16	0.08
24 cores of CPU	65.74	176.65	0.37
V100 GPU	407.62	173.63	2.34
1 FPGA kernel	74.29	45.61	1.63
2 FPGA kernels	146.94	52.47	2.80
4 FPGA kernels	289.02	71.98	4.02

Table 1: Performance and energy efficiency comparison of FPGA kernels [16].

One of our FPGA kernels draws 45.61 Watts (the FPGA idle with the bitstream loaded draws 39 Watts), and whilst the energy efficiency of 1.63 GFLOPS/Watt of a single kernel is significantly higher than the CPU, it is somewhat disappointing when compared against the GPU.

However, the advantages of FPGAs start to become more apparent as we scale the number of kernels. We can fit up to four of our kernels on the U280, and at this configuration we achieve

289 GFLOPS. This is over four times the performance of the 24 core CPU, and 71% of the performance of the V100 GPU. The energy consumption of four kernels is 72 Watts and it can be observed that, on average, adding an extra kernel requires approximately an additional 7 Watts, with a performance increase close to 74 GFLOPS per kernel. With four kernels, the energy efficiency is over 4 GFLOPS/Watt, which is significantly higher than that of the GPU. Therefore, whilst best performance of the FPGA vs GPU still favours the GPU (although it is a tough test), energy-efficiency-wise there are significant advantages of using FPGAs.

2.5.2 HPC architectures at CINECA and E4

Nekbone has been compiled on four different classes of system reported below for both compiler-specific and compiler-independent tuning. The four systems are as follows:

Galileo cluster (CINECA).

1000 Intel Broadwell nodes (2x18-core Intel Xeon), and OmniPath interconnection, plus 60 nodes equipped with K80 Nvidia accelerators and 2 with V100 Nvidia accelerators.

Marconi A3 partition (CINECA):

Lenovo NeXtScale platform, featuring Intel Xeon Skylake (SKL) processors, with a peak performance of about 20 PFLOP/s.

Marconi100 (CINECA)

980 nodes; each node is equipped with 2x16 cores IBM POWER9 AC922 plus 4 NVIDIA Volta V100 GPUs, all connected with a high-speed internal network Mellanox InfiniBand EDR DragonFly+.

ARMIDA (E4)

ARM Infrastructure for the Development of Applications cluster, located at E4 Computer Engineering's premises, with 8 Marvell TX2 compute node, each with 64 Marvell TX2 cores and a number of Nvidia Tesla V100.

Nekbone has run with 100 CG (Conjugate Gradient) iterations, polynomial degree 9 and for 10 GLL points in each dimension. The reference test is "example2" from the official repository [17]. It runs without the multigrid preconditioner and without user-provided decompositions of the processor counts and the elements

Figure 3 below shows the weak scaling, by keeping constant the number of elements per core (128), with the different architectures listed in the legend. Using a patch about MPI tag misuse in the code, Nekbone ran well over 152,160 cores on Marconi A3, with a peak performance of 35,9 TFLOP/s, using OpenMPI (3.0).

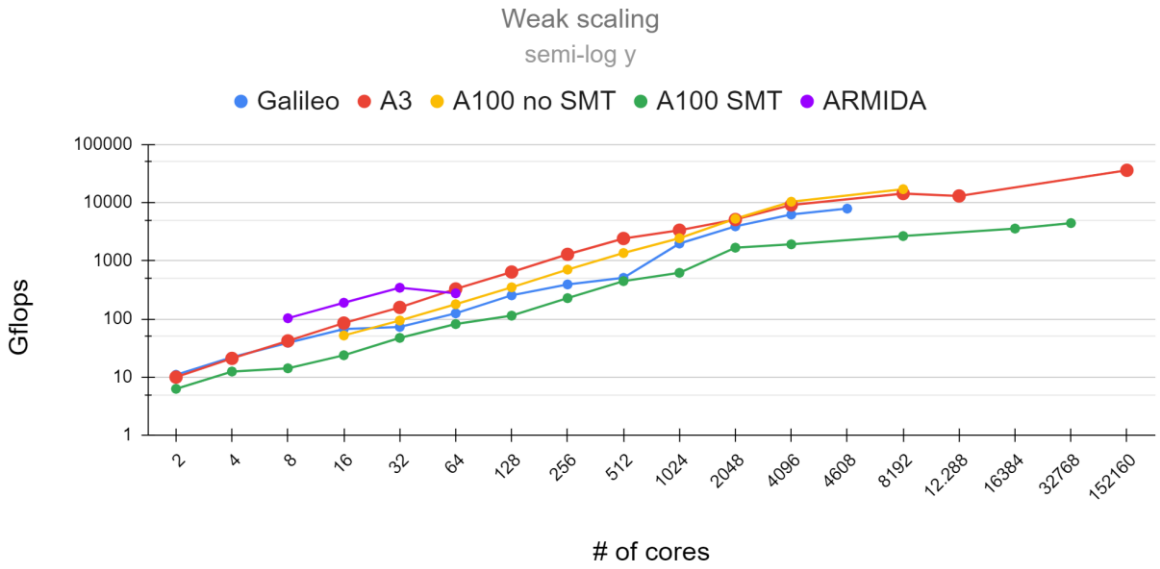


Figure 3: Nekbone weak scaling on various CPU clusters.

Figure 4 below shows the weak scaling on Marconi100 using the full machine. The best performance has been achieved recently at 835 TFLOP/s (near Petascaling performance) in OpenACC configuration, out of a total of 3792 GPUs, with a weak scaling on a full machine.

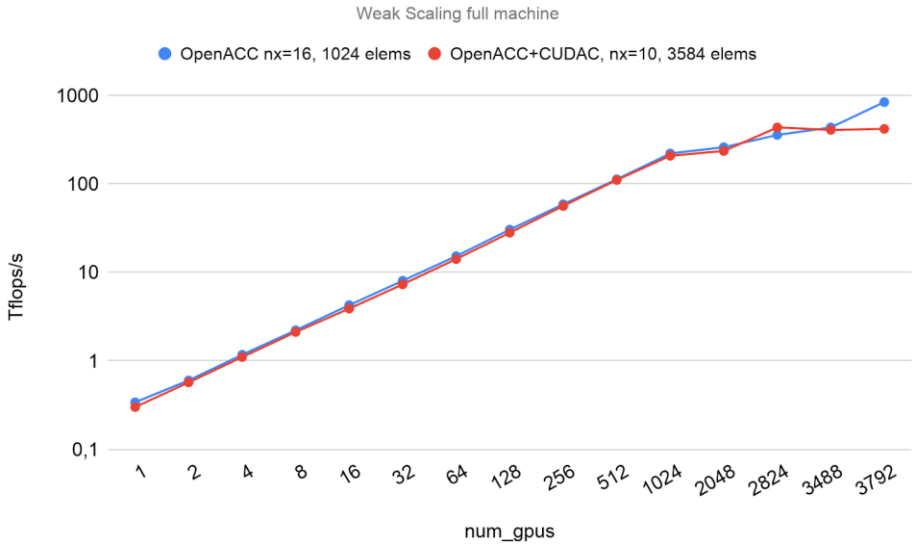


Figure 4: Weak scaling of Nekbone on Marconi100.

2.5.3 SX-Aurora at HLRS (KTH)

KTH has started to port and tune the entire spectral element code Nek5000 to the SX-Aurora TSUBASA system at HLRS. Compared to heterogeneous computing platforms, SX-Aurora offers a friendlier programming model, with a native execution mode, allowing a developer to use the full potential of the system without having to deal with the complexities of heterogeneous systems. However, good performance is only achieved if the code vectorizes well.

The experience of porting and tuning Nek5000 on GPUs helped us to formulate suitable loop transformations for increased vectorization and work per iteration throughout the code. In key, compute-intensive kernels, the transformations achieved 40% of the theoretical peak performance of a single SX-Aurora core. Figure 5 shows the performance of the Nekbone mini-app running on three different hardware architectures, SX-Aurora, two Intel E5-2698v3 CPUs and a Nvidia P100 GPU. Using all eight cores, Nekbone achieved close to 10% of the SX-Aurora's peak performance compared to the 5.5% of peak performance achieved for the Nvidia P100 GPU. The porting and tuning efforts will be presented in a paper accepted for HPCAsia 2021 [18] .

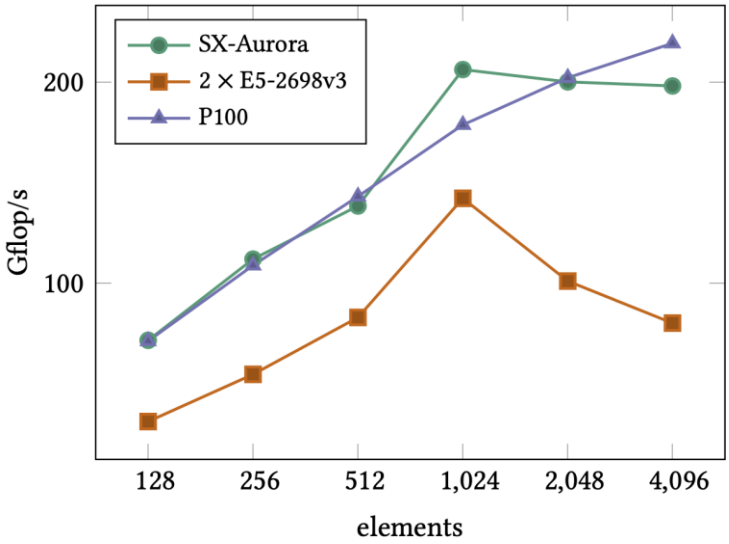


Figure 5: Performance of Nekbone running on three different hardware architectures.

2.6 ScaLAPACK/RWTH

The target ScaLAPACK routine, namely the singular value decomposition routine *pdgesvd*, has yet to be developed into a mini-app, and is scheduled to occur over the coming months, along with extending the functionality of the DMD algorithm (See T4.3 Data Analysis, Section 4.3). Once the mini-app will be ready, we will start with profiling analysis and performance measurements on the various systems and architectures available at CINECA, starting with the Intel-based Tier-1 cluster

2.7 Summary of Co-Design Activity

The WP-transversal co-design working group and the co-design task (Task4.1), have contributed to EXCELLERAT's Advanced Services. On the software co-design side, based on the reference codes and other relevant applications and libraries of the engineering workflow, the operations, kernels and algorithmic features are characterized that are common and widely used in engineering applications and which demand large amounts of computational time. On the hardware side we are cooperating via its interest groups with original equipment manufacturers and system integrators considering the complete hardware bandwidth from standard x86_64 architectures to ARM CPUs, GPUs, NEC Vector processors down to FPGAs.

Thanks to our efforts over the last year, we now have an agreed methodology to follow, wherein the client's application (or mini-app) are ported to our collection of cutting edge HPC platforms, and the required code-adaptations are then fed back to the client.

To date, we have exercised our nascent co-design service using CoE members. As such, our clients have been our own Reference Applications owners. Five applications have been considered for our co-design service, where two of these have mini-app versions, and a third mini-app is on the horizon. Four of the applications have now been ported to a number of novel machines, including CPU clusters, GPU clusters, and a FPGA cluster, and initial results have been highly promising. The act of porting was either performed by the owners themselves or, for two of the applications, by CoE members at other core partner sites. For the latter case, the required code-adaptations were fed back to the code owners where.

Finally, it is important to note that, given the success of our period of testing, we are proud to announce that the EXCELLERAT CoE now provides consulting for software and systems co-design. This service is now live, and available within the Co-Design Engineering Software- and System-Design service. See the *Reports on the Service Portfolio*, deliverables D4.2 and D1.6 for a more detailed description.

3 Task 4.2: Visualization

To focus I/O on a manageable level and to supervise simulations during runtime, the main goal of this task is the enablement of in-situ post-processing capabilities in selected applications.

One tool chosen for this purpose is Vistle [19] for which two in-situ interfaces have been developed. The first one is the LibSim interface used by VisIt [20] as outlined in D4.2. The second interface is obtained via the SENSEI [21] framework. In D4.2 we described an approach made using Catalyst [4], but this was abandoned. This decision was made because SENSEI follows a similar approach as Catalyst in terms of the simulation data adapter but also provides an interface for post-processing backends like Vistle and is, therefore, more flexible. In this project, a backend adapter for Vistle was developed.

While Vistle is focused on interactive visualization in 3D virtual environments, the other post-processing tool developed in this project is a High-Performance Computing analysis tool (PAAKAT). This tool allows a straightforward real-time handling of data arisen from simulations with pre-configured algorithms based on the Visualization Toolkit (VTK) [22].

3.1 Interactive in-situ visualization with Vistle

Vistle implements a modular architecture, where every data/-source, -filter or -sink in the post-processing pipeline is represented as a Vistle module. In multi-process mode, these modules run as separate processes, that communicate and share their data via shared memory (SHM). Therefore, to retrieve data from simulations, the data must be copied to a Vistle-controlled SHM segment.

In single-process mode all modules run in a separate thread of the single main process. In this mode data objects are passed on directly via pointers. This would theoretically allow direct usage of simulation data, but practically Vistle's internal data representation is not yet suitable for input of outside of Vistle allocated data. Also, these threads use MPI independently, MPI_Init_thread with MPI_THREAD_MULTIPLE is required. Running Vistle, in single-process mode, coupled with a simulation, requires the simulation to initialize its MPI environment accordingly.

A connected simulation is represented in the Vistle pipeline as a regular Vistle module, as shown in Figure 6. These modules only request simulation data for the connected data ports and in configurable intervals. Therefore, only the requested data is converted and passed to the Vistle pipeline. These connections, the following pipeline modules, as well as the in situ-modules' parameters can be changed during the run-time of the simulation to provide maximal flexibility. The cost of this flexibility is the necessity of transforming and copying the simulation data into Vistle's representation. While the computational overhead is expected to be rather small, the memory overhead can be huge, especially if multiple time steps are kept for visualization.

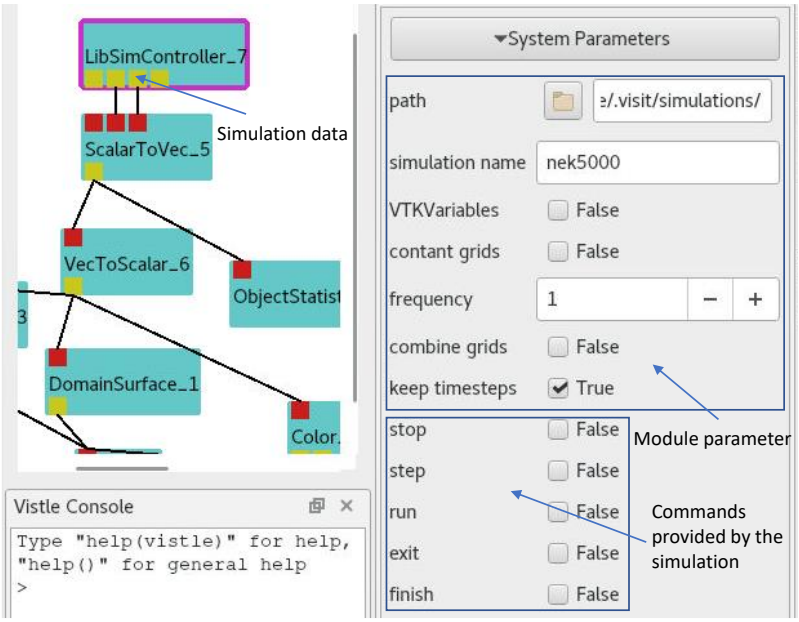


Figure 6: Connected LibSimController-module.

3.1.1 LibSim interface for Vistle

The LibSim interface consist of a small library that is statically linked to simulation codes. The simulation uses this library to pass data retrieving callbacks to the post-processing back-end. The interface passes around raw data-array pointers and therefore avoids VTK. Once such a simulation starts, it uses a connection socked that waits for the back-end to connect. After the connection, the static library dynamically loads a run-time library. This is where Vistle intervenes through replacing VisIt's libsimV2 runtime library with an own implementation. This dynamically linked library manages the communication with the Vistle module and the data conversion from LibSim to Vistle. A brief user guide on how to use Vistle's LibSim interface is presented in the Appendix of this deliverable (see Section 9.2.1).

3.1.2 Vistle post-processing backend for SENSEI

SENSEI provides an interface, that post-processing back-ends can adapt, to connect to SENSEI instrumented simulations. Which back-ends are used can be configured at the start of a simulation via SENSEI's configurable analysis adapter. The SENSEI interface adapted from Vistle converts the VTK objects provided by SENSEI simulations to Vistle-objects, which are then inserted in the Vistle pipeline. While the LibSim interface allows simulations to register arbitrary commands this interface only features the basic run and pause commands. A brief user guide on how to use Vistle's SENSEI interface is presented in the Appendix of this deliverable (see Section 9.2.2).

3.2 In-situ visualization with PAAKAT

The PAAKAT ("Looking at") library has been designed as an HPC tool which encourages scalability and portability of in-situ analysis in large-scale simulations. The emphasis is on reducing such simulations' output data during run-time by using algorithms already available in VTK. The main difference regarding the great deal of effort made to develop software specialized on the solution of in-situ visualization and analysis is related to the fact that PAAKAT encourages scalability and portability. This has been done by focusing on data arisen from VTK filters while it obviates rendering in the ParaView [23] source code (version 5.6). Modifications performed in ParaView encourages the use of the C++ VTK API. As a consequence of these modifications, filters must be implemented by using C++ instead of the Python [24] scripts created by ParaView.

Using the whole ParaView framework, compilation times became a real issue. Tests on four nodes of MareNostrum 3 (Barcelona Supercomputing Center, Spain) [25] with a total of 64 cores resulted in a compilation time of about one and a half hours. By obviating rendering and therefore avoiding the need of big third-party libraries like Python and/or OpenGL [26], the compilation time was reduced to only nine minutes.

A list of the compiler options used to achieve this are show in Figure 7. These options have been successfully used in two supercomputers, Beskow (KTH Royal Institute of Technology, Sweden) and Nord III (Barcelona Supercomputing Center, Spain). As future work, more computer systems and compilers must be tried.

```
cmake PATH/TO/PARAVIEW5.6/SOURCES  \
-DCMAKE_INSTALL_PREFIX=EXECS01 \
-DPARAVIEW_BUILD_QT_GUI=OFF \
-DCMAKE_CXX_COMPILER=mpicxx  \
-DCMAKE_C_COMPILER=mpicc \
-DCMAKE_Fortran_COMPILER=mpif90 \
-DPARAVIEW_USE_ICE_T=OFF \
-DPARAVIEW_USE_MPI=ON \
-DBUILD_SHARED_LIBS=OFF \
-DVTK_Group_ParaViewRendering=OFF \
-DVTK_USE_X=OFF \
-DVTK_OPENGL_HAS_OSMESA=OFF \
-DVTK_OPENGL_HAS_EGL=OFF \
-DModule_vtkIOExport=OFF \
-DVTK_BUILD_ALL_MODULES_FOR_TESTS=OFF \
-DVTK_Group_Rendering=OFF \
-DVTK_Group_StandAlone=OFF \
-DVTK_Group_MPI=ON \
-DModule_vtkCommonCore=ON \
-DModule_vtkFiltersGeneral=ON \
-DVTK_RENDERING_BACKEND=None \
-DPARAVIEW_ENABLE_VTK_MODULES_AS_NEEDED=FALSE \
-DModule_vtkVTKm=ON \
-DModule_vtkAcceleratorsVTKm=ON \
-DModule_vtkPVVTKExtensionsDefault=ON \
-DPARAVIEW_ENABLE_COMMANDLINE_TOOLS=OFF \
-DPARAVIEW_CURRENT_CS_MODULES=
```

Figure 7: Compiler options for modified ParaView 5.6.

3.2.1 MPMD-VTK example

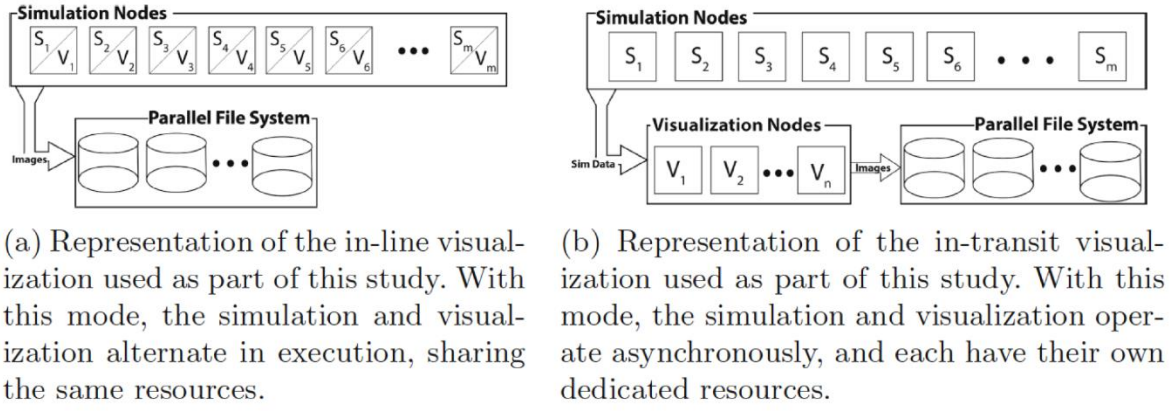
In this example two codes are run by using the Multiple-Program, Multiple-Data (MPMD) programming model while an interpolation procedure is performed.

A given source mesh is used to perform a turbulent simulation, while a principal component analysis (PCA) is carried out through a destination mesh. Input files for this problem have been supplied by Christian Gscheidle (Fraunhofer SCAI) as part of the Data Analytics task.

The problem can be solved in three main steps. First, interpolation is performed, then the interpolated data is extracted from ParaView, and thereafter used it as input data in the PCA analysis. By means of an appropriately compiled ParaView, the procedure described above can be performed completely using Python (*paraview.simple.Resample.WithDataset*,

paraview.simple.ProgrammableFilter, and *sklearn.decomposition.PCA* for interpolation, extraction, and machine learning analysis, respectively).

Now, the parallel in-situ execution of this procedure is usually performed by using the Single-Program, Multiple-Data (SPMD) programming model in which the same processors execute numerical simulation and visualization in a staggered way. This means that they are executed one after the other. Another option is to execute a MPMD programming model where different processors are given to simulation and visualization. In this case, simulation and visualization are independent, but extra communication between sets of processors is necessary. While the MPMD model is widely used in numerical simulations of multi-physics problems, its effect on in-situ visualization problems has been recently studied (See Figure 8). In the problem that is being tested here, the usage of a MPMD execution could encourage the performance and load balance of the entire parallel execution (numerical simulations and in-situ). Additionally, compilation of the codes can be simplified, since codes are independent of each other. This means that the turbulent flow problem could be simulated using whichever parallel code (with or without in-situ instrumentation) while the analysis could be performed by means of either C++ or python, and even using either serial or parallel codes.



(a) Representation of the in-line visualization used as part of this study. With this mode, the simulation and visualization alternate in execution, sharing the same resources.

(b) Representation of the in-transit visualization used as part of this study. With this mode, the simulation and visualization operate asynchronously, and each have their own dedicated resources.

Figure 8: Parallel programming models. (a) Single-Program, Multiple-Data (SPMD) or in-line visualization. (b) Multiple-Program, Multiple-Data (MPMD) or in-transit visualization [27].

As mentioned above, another interesting point is related to the load balance. Figure 9 shows the number of cells and dimensions of the given meshes. By using a standard domain decomposition method for the distribution of the parallel work in the turbulent flow problem, it is relatively easy to conclude that during the execution of the in-situ stage most of the processors could be idle when a SPMD execution model is used.

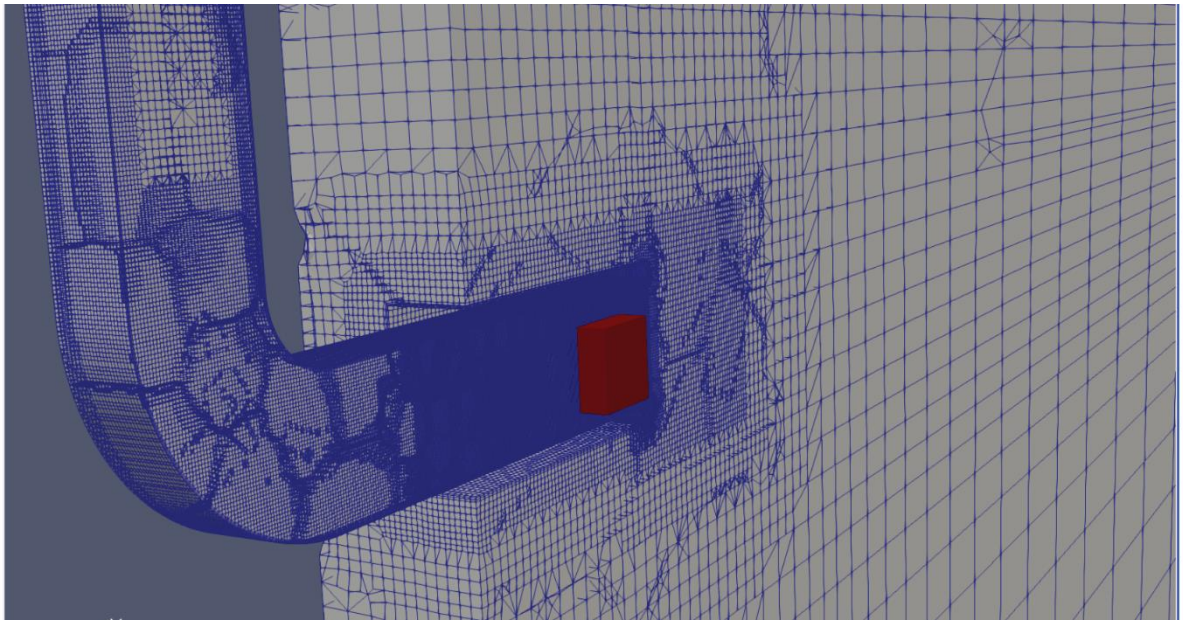


Figure 9: Source mesh (blue and grey) of 4.67e6 cells and volume of 4.43x4.09x0.22 cm^3. Destination mesh (red) of 1.25e5 cells and volume of 0.05x0.05x0.05 cm^3.

In order to overcome this drawback, an example using the MPMD model has been prepared (see Figure 10) The setup consists of two different codes, one for the destination mesh and another for the source mesh. The first one reads the destination mesh and sends its points to the source-code. The second code reads the source mesh and performs interpolations with data received from the destination code. Initially both codes are executed at the same time, and once meshes are read, the destination code sends its points towards the source code. This one performs the interpolations and sends back the results to the destination code. In the future, destination code will use the received data to perform the corresponding machine learning analysis.

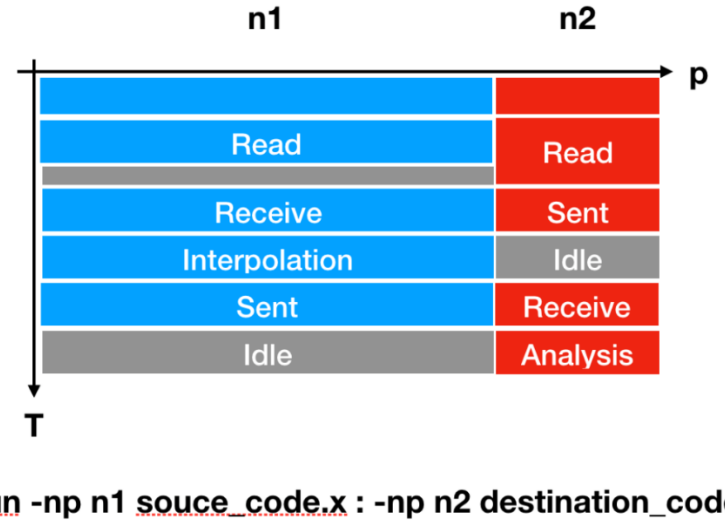


Figure 10: MPMD execution model of source code and destination code through n1 + n2 processors p.

3.3 Summary of Visualization Activities

The main focus of the work in task 4.2 has been to put on the development of post-processing tools and workflows for exascale engineering simulations. Next to consulting in visualization topics and training offerings, the main effort of the visualization task is the development of scalable in-situ post-processing tools. New things we bring to the table are interactive in-situ visualization in virtual environments and performant in-situ analysis of large-scale simulations.

With the presented in-situ approaches it is possible to easily couple Vistle with a wide variety of simulations without the need to implement a special in-situ interface. The drawback is that the transformation of data has to be done twice, once from the simulation's representation to LibSim/SENSEI format and then to Vistle's format. Because Vistle's data objects are designed to control the lifetime of their data, even in the case where no transformation would be needed, the data must be copied into Vistle's object representations (in multi-process mode to SHM, but also in single process mode). On the one hand this can cause a huge memory overhead, but on the other hand it allows to interact with the data independently of the simulation.

So far, the core-code that has been successfully run in-situ with Vistle was Nek5000 by using its LibSim interface. In the next phase of the project more core codes will be integrated, and measurements of the overheads will be made.

Similarly, PAAKAT has shown encouraging results for in-situ analysis of large-scale simulations. The simple and flexible API made integration into both Nek5000 and Alya straightforward, demonstrating the potential of PAAKAT as a general in-situ toolkit.

4 Task 4.3: Data analytics

4.1 In-Situ Data Analytics

4.1.1 Software Architecture

Fraunhofer has continued the development of a software framework for the purpose of in-situ analysis of CFD data. Some key requirements have been identified which are considered during the implementation:

- Online and offline application of the toolbox
- Simple integration with existing ML and UQ libraries
- Efficient back-end based on (parallel) linear algebra libraries
- Connection to standard in-situ interfaces and data readers
- Interactive user-interface based on a client/server setup

The overall concept of the selected architecture is shown in Figure 11. For the purpose of a fast and flexible development and validation of algorithms, most existing Machine Learning libraries provide a Python API and allow an interactive usage through ipython or jupyter servers [28]. In order to take advantage of existing libraries, this is also a fundamental design strategy for our software toolbox. The core API follows the definition by scikit-learn [29] and thus enhances a simple integration of our code with external methods into a single processing pipeline.

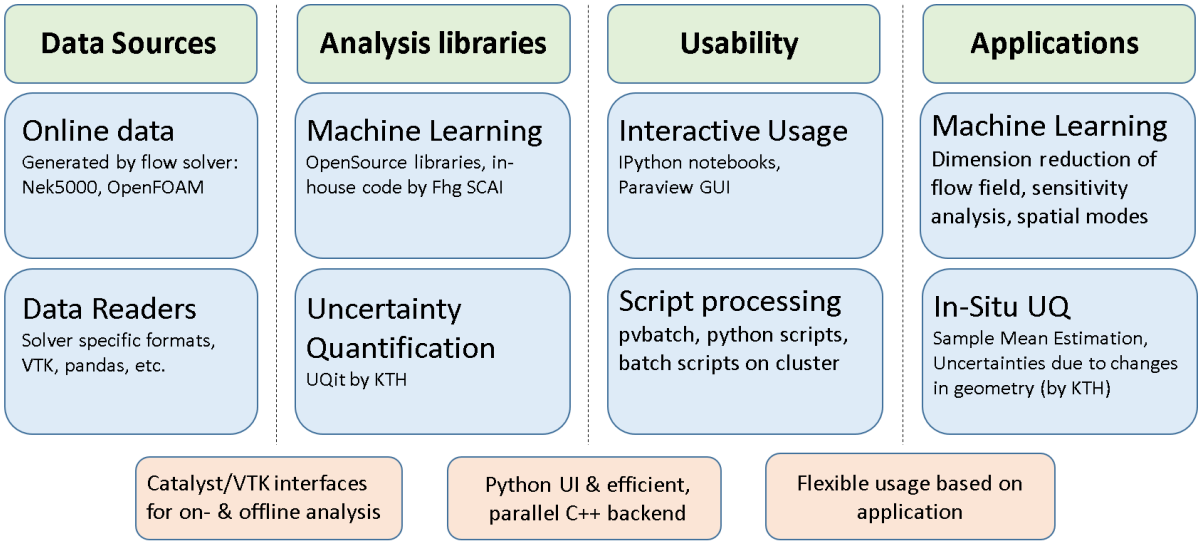


Figure 11: Software architecture for in-situ data analysis [30].

Besides flexibility, efficient implementations of the algorithms are a major requirement in the context of large-scale data analytics. Therefore, we keep the python layer as thin as possible and build on existing math libraries, e.g. OpenBLAS or MKL, for heavy computations. To further increase efficiency, a high-level parallelization is set up via mpi4py [31]. For low level parallelizations that rely on more intense communications, efforts have been put into the

implementation of a python wrapper for the Elemental parallel linear algebra library [32] for selected algorithms. Further efforts will be spent on this work in the reaming time of the project.

For in-situ data analysis, some of the methods have been wrapped as VTK [22] filters. In this way, they can be executed online in a Catalyst [33] pipeline or offline as ParaView [23] plugins. The latter also enables the import into a python environment that provides interactive access to all VTK functionality and integration with other analytics libraries in a client/server setup.

4.1.2 Applications

In collaboration with KTH, UQit (see next section) has been applied to the three-dimensional flow around an ellipse generated by NEK5000 in an in-situ way. Thereby, the sample mean of the entire high-dimensional velocity field and its corresponding confidence intervals were computed during the run time of the simulation. A slice through the mid-plane of the ellipse is shown in Figure 12 together with the 95% confidence interval.

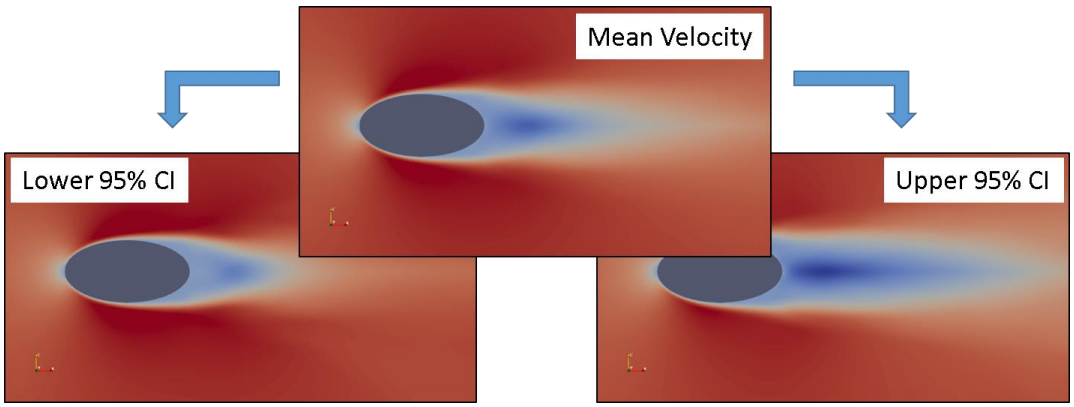


Figure 12: Mean velocity field with corresponding 95% CI.

Results are constantly updated throughout the simulation. Therefore, the confidence interval for a given quantity of interest can be used as a stop criterion for the simulation once it is below a predefined threshold.

In a second example, the Principal Component Analysis (PCA) of the velocity field was computed during the simulation based on a batch approach. By applying this method, several snapshots of the flow field are cached into memory and processed together to update the PCA regularly.

Besides a reduction of the data in- and output during and after the simulation, a further benefit of the method lies in earlier availability of the results. This can be exploited to generate a low-dimensional representation of the flow field during run time. To demonstrate a possible application, a sensitivity analysis of the flow field was performed to find correlation with a quantity of interest. In a further step, these results can be used as a criterion for Adaptive Mesh Refinement (AMR).

In the remaining time of the project, we will focus on the development of further in-situ data analytics methods. Besides the sensitivity analysis, additional applications for Machine Learning, such as building data-driven surrogates, will be investigated.

4.2 Uncertainty Quantification

The development of UQit, a Python package for uncertainty quantification (UQ) in CFD (computational fluid dynamics) has reached to a good level. The first version of this open-source package has been recently released along with a journal paper submitted to JOSS (Journal of Open-Source Software) [30]. UQit is designed to be non-intrusively linked to any CFD solver, conditioned on having appropriate interfaces. From the programming point of view, the design is flexible so that UQit can be efficiently extended in future. An extensive documentation of UQit has been prepared which covers a short theory and detailed implementation of each UQ technique, and also provides several examples and notebooks. From this aspect, UQit can be used for pedagogical purposes, as well. In the first release several features are included, which are shortly reviewed below.

Uncertainty propagation or UQ forward problem is performed through standard and probabilistic polynomial chaos expansion (PCE). The probabilistic version is a novel feature that is provided in UQit. The construction of PCE is flexible in terms of the types of samples, truncation scheme, dimension of the parameters, and the method for estimating the unknown coefficients. Regarding the latter, the possibility of using the compressed sensing method has been provided. Global Sensitivity Analysis is performed through computing main, interactive, and total Sobol indices. There is no limitation regarding the number of samples and how the parameters are (randomly) distributed. Sampling, both training and test, can be carried out using several stochastic and spectral methods. For construction of the surrogates which are required for efficiently studying different types of UQ problems, options such as Lagrange interpolation, PCE, and more importantly, Gaussian process regression (GPR) are considered. In order to handle the typical problems arising in CFD, we have considered GPR with both observation-dependent (heteroscedastic) and -independent (homoscedastic) noise structure.

The theoretical development of UQit and its application to computing validation and verification (V&V) metrics in CFD are described in a submitted journal paper that has been under review since July 2020 [34]. Besides this, another paper has been produced (not submitted yet) in which the UQ framework for V&V is applied to simulations of Nek5000 and OpenFOAM.

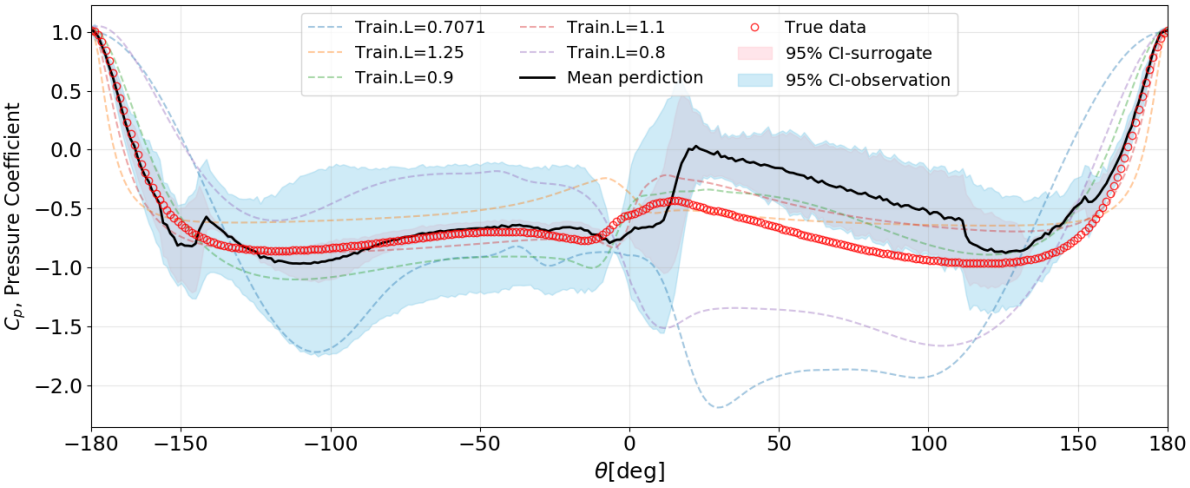


Figure 13: Distribution of the pressure coefficient over the ellipse boundary. The diameter of the ellipse is assumed to be uncertain. Using a set of training simulations, the pressure coefficient for an unseen geometry is predicted and compared to the true simulated data.

In a collaboration between KTH and Fraunhofer, we have successfully linked (offline) UQit to the simulations of Nek5000. For the flow over an ellipse, the simulation data were extracted by Paraview. The quantities of interest (QoIs) can, for instance, be pressure and friction coefficients over the boundary of the ellipse, or velocity/pressure signals at any point inside the domain. A computer experiment was designed in which the diameters of the ellipse were assumed to be uncertain conditioned on keeping the area of the ellipse fixed for all realizations. Using UQit, we could show how to estimate the uncertainties in the flow QoIs due to the variation of the ellipse shape. In addition to this, we could make predictions for the QoIs and their uncertainties, for the geometries which were not actually simulated, see Figure 12. This has been done using the Gaussian process regression. For the same set of computer experiments, we estimated the propagated uncertainty and also made predictions for the POD (proper orthogonal decomposition) modes computed in-situ from the flow over ellipses of different geometries.

In another direction of developing UQ techniques for large-scale CFD simulations, we have made a good progress in theoretical development of the techniques for reliable estimation of uncertainty in time-averaged quantities. Some of the developed features have been already tested for C1U1 (KTH) use case that is a 3D wing with round tip. The reliable estimation of uncertainties due to finite time-averaging is crucial considering the limitation in computational resources for wall-bounded turbulent flows at moderate to high Reynolds numbers. In this regard, through another collaboration between KTH and Fraunhofer, we have attempted the in-situ estimation of uncertainties due to time-averaging. The initial results are promising, and the work is in progress. The interface between the flow solver and the UQ-suite is based on the VTK (visualization toolkit) packages. In connection to the described projects, several research directions are planned to be pursued in the remaining time of the EXCELLERAT project which include the followings: (a) further development of the theories and implementation of the derived methods for reliable estimation of uncertainties in time-averaged flow quantities, (b) further development and test of the in-situ estimation of the uncertainties due to time averaging

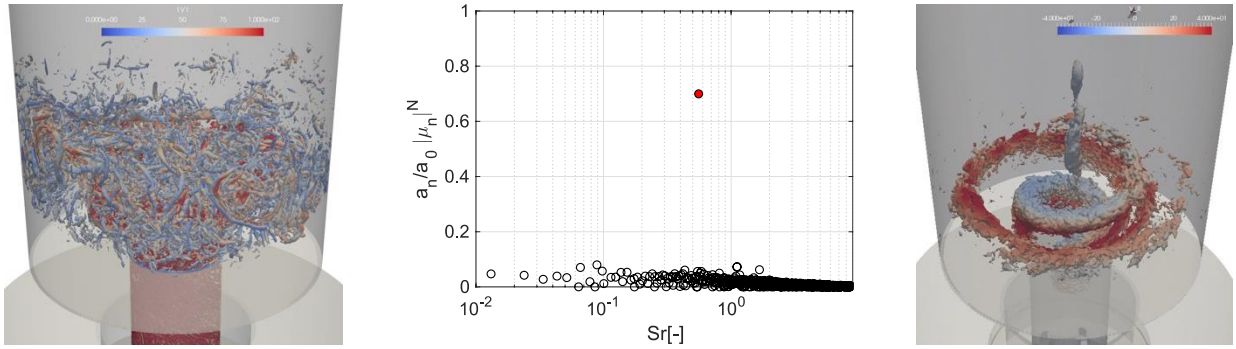
(KTH-Fraunhofer collaboration), and (c) further theoretical development and numerical investigation of a new UQ-based strategy for adaptive mesh refinement (AMR) in Nek5000. The idea of the latter is novel, and at this stage we are studying its feasibility.

4.3 Calculation of Modal Decompositions

Highly accurate, turbulence scale resolving simulations, i.e., large eddy simulations and direct numerical simulations, have become indispensable for scientific and industrial applications. Due to the multi-scale character of the flow field with locally mixed periodic and stochastic flow features, the identification of coherent flow phenomena leading to an excitation of, e.g., structural modes is not straightforward. A sophisticated approach to detect dynamic phenomena in the simulation data is a reduced-order analysis based on dynamic mode decomposition (DMD) or proper orthogonal decomposition (POD).

After the modal decomposition tools have been optimized in the first year of EXCELLERAT to efficiently perform modal decompositions in parallel on large data volumes, the tools have been utilized to investigate the hydrodynamic instability in a swirl-stabilized combustor of the Alya-C2U1 use case provided by BSC that is described in D2.2 (Report on Reference Applications Outcomes) [35]. Based on the flexible modular reader interface of the decomposition tool, a new parallel reader has been implemented base on the BSC Alya MPIO IO library to efficiently read the simulation data in the native Alya format.

The flow field of the investigated combustor exhibits a self-excited flow oscillation known as a precessing vortex core (PVC) at a dimensionless Strouhal Number of Sr=0.55, which can lead to inefficient fuel consumption, high level of noise and eventually combustion hardware damage. To analyse the dynamics of the PVC, DMD is used to extract the large-scale coherent motion from the turbulent flow field characterized by a manifold of different spatial and temporal scales shown in Figure 14 (a). The instantaneous flow field of the turbulent flame is visualized by an iso-surface of the Q-criterion coloured by the absolute velocity. The DMD analysis is performed on the three-dimensional velocity and pressure field using 2000 snapshots. The resulting spectrum of the DMD, showing the amplitude of each mode as a function of the dimensionless frequency is given in Figure 14 (b). One dominant Mode, marked by a red dot, at Sr=0.55 matching the dimensionless frequency of the PVC is clearly visible. An instant of the temporal reconstruction of the extracted DMD mode superimposed on the temporal averaged flow field showing the extracted PVC vortex is illustrated in Figure 14 (c). It shows the iso-surface of the Q-criterion coloured by the radial velocity.



(a) (b) (c)

Figure 14: DMD analysis performed on the flow field of a turbulent flame. (a) Instantaneous flow field, (b) Spectrum of the DMD, (c) Reconstruction of the dominant DMD-mode.

Since the modal decomposition toolkit developed in EXCELLERAT addresses a broad user community having different backgrounds and expertise in HPC applications, a user-friendly implementation allowing an efficient workflow is mandatory. To achieve this requirement, a server/client structure has been implemented which is shown in Figure 15. Using this structure, the actual modal decomposition is done on the server running in parallel on the HPC cluster which is connected via TCP with the graphical user interface (client) running on the local machine. To efficiently visualize the extracted modes and reconstructed flow fields without writing large amounts of data to disk, the modal decomposition server can be connected to a ParaView server/client configuration via Catalyst enabling in-situ visualization.

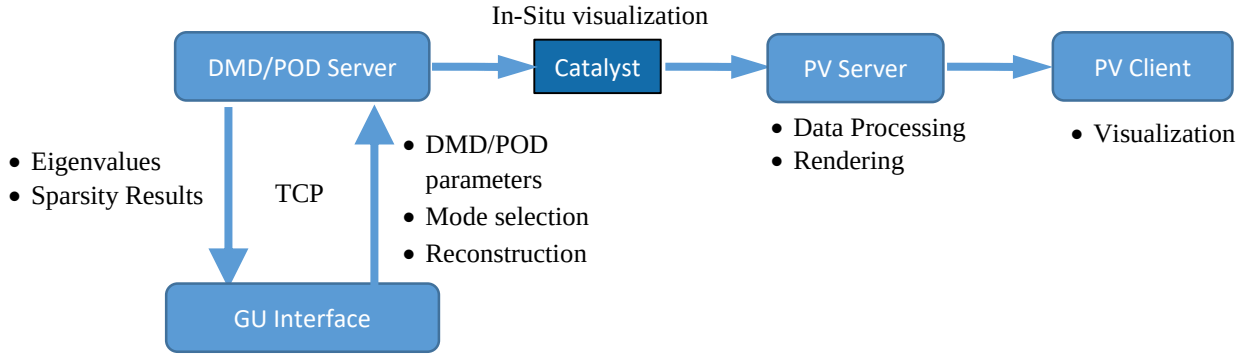


Figure 15: Client/Server structure.

4.4 Post-Processing TPLS Simulations

During the second year of EXCELLERAT, UEDIN undertook an investigation into the viability of introducing in-situ data analysis in the TPLS simulation code.

TPLS simulates two-phase turbulent flow and the version handed over to UEDIN to prepare for exascale outputs two forms of data files: Snapshot and Restart files. Both files were in ASCII and were written serially by the master processor. The author of the code assumed that the entire dataset could fit inside the memory of a single MPI process. Clearly, to run efficiently on exascale machines, much larger simulations are required which will be too large for a single MPI I/O process. To overcome this, it was agreed that UEDIN would ensure efficient I/O

performance in the future, by transforming the I/O from ASCII to a binary format and parallelising the I/O routines. This work was performed under Task 4.4: Data Management and is described later in this document. The Snapshot files contained the 5 or 6 parameters for each of the points in the 3D mesh, where the number of parameters was hard-wired into the source code, thus end users may have to recompile and rerun simulations. The Restart files were larger in size, as they contained the same information as the snapshot files, but also further information required to enable the simulation to restart from a previous simulation. Because the number of parameters is hard-wired into the source code, the end user needs to recompile and rerun the simulation each time the post-processing data analysis was found to require more data. This too has been addressed under Task 4.4 and is described later in this document.

The data analysis carried out by the TPLS owners involves post-processing the snapshot files using Matlab, wherein some basic calculations are performed before a plot is created for each snapshot file. These plots were saved to file and displayed on the screen thereby creating an animation viewed by the end user.

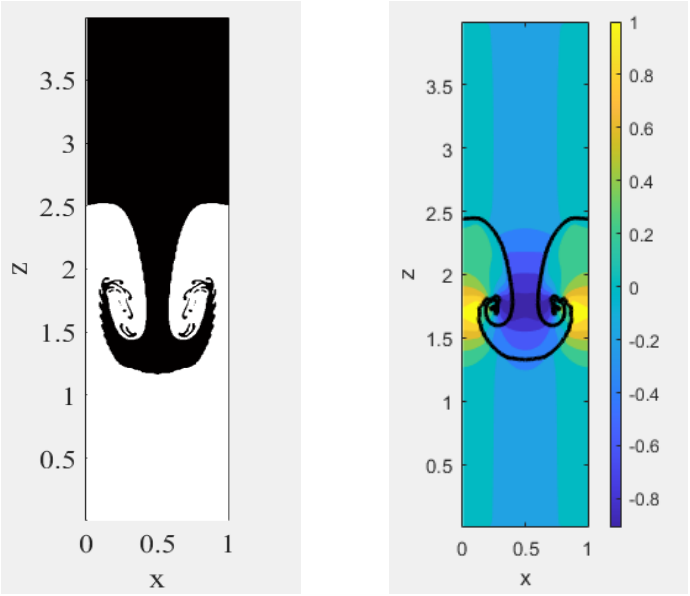


Figure 16: Two images created by the two Matlab post-processing routines.

UEDIN approached the TPLS owners with the idea that post-processing data analysis could be done inside the code in-situ, where one core per node would be removed from the pool of cores employed for the simulation. This collection of cores, one per node, would then be employed to perform the basic calculations of post-processing using Fortan90 and MPI, the language and communications library of TPLS, and the graphics files would be written directly using Fortran90. This would involve changing the format of the graphic files from the Matlab *.fig* format to *.png* or more likely *.ppm*. Moreover, the Matlab routines would not only need to be rewritten in Fortran90 (or C) but also parallelised to compute on the distributed data. This data would be gathered from the running simulation using an interface such as Catalyst from ParaView, and libSim from VisIT, depending on their suitability.

The Matlab routines and example data files, were received and, after some effort, the post-processing environment was recreated on a laptop locally at UEDIN. The port was not exact as, despite the images appearing correct to the eye, the resultant binary *.fig* files differed. Despite both runs performed on Windows 64-bit OS, they were produced using different versions of Matlab (UEDIN used R2020a in 2020, whilst the results employed for comparison were generated almost 5 years ago).

During the investigation to convert the Matlab *.m* files to Fotran90, and then parallelise them with MPI, it became quite clear that the amount of calculation was simply too small to warrant parallelising the calculation. Such work would result in a highly inefficient use of the resources due to extremely poor load-balance: the snapshots graphic files are created once every 500 timesteps, thus the in-situ data analysis cores are mostly waiting to be invoked; when invoked each core would then compute almost nothing, then these cores would spend the majority of their active time writing the graphic files using a parallel I/O harness and, given that communication networks are typically a shared resource, this traffic would occupy the same network employed by the TPLS simulation itself and thus slow down the ongoing simulation.

As described in a following Section, the TPLS I/O routines now employ PETSc I/O routines, writing binary HDF5 files, and the resultant snapshot files are much smaller and created much faster. The snapshot files now hold all parameters that the end user may wish to post-process, removing the need to recompile and rerun simulations.

Given the improved I/O and the projected inefficiencies of running the post-processing in-situ, the owners of TPLS agreed the best use of the remaining UEDIN T4.3 effort would be to convert their post-processing Matlab routines to read HDF5 rather than the ASCII files, thereby alleviating disk space required to store the snapshot files, and to allow users to decide what is plotted after the simulation and not during compilation.

It has been found that the given example ASCII files employ an out-of-date data structure, thus the Matlab scripts require alteration before work can start to switch to HDF5. Moreover, a new numerical method has been introduced to TPLS, where simulations may now employ either the existing Level Set Method or a new Diffuse Interface Method, where this new method employs a different parameter range. As such, new Matlab scripts are to be developed to cater for all these possibilities.

4.5 Summary of Data Analytics Activities

The main focus of the work in task 4.3 has been put on the theoretical development and enhancement of dedicated software tools for data analytics of CFD simulations, namely in the following areas:

- In-Situ data analysis and simulation monitoring
- Uncertainty quantification
- Modal decompositions of transient flow data
- Enhancements on TLPS post-processing capabilities

An in-situ software toolbox was developed by Fraunhofer as a basis for several data analysis algorithms that are executed during the simulation for different purposes, such as simulation

monitoring, sampling mean error estimations or building data-driven surrogates of the flow. Gained knowledge during the process and documentation material will be exploited in training activities as well as consulting services related to “Data Analytics for Engineering using Machine Learning”.

Theoretical development of various UQ techniques for CFD applications have been pursued by KTH and implemented in UQit, a dedicated python package for UQ. Theoretical results and details on the implementation have been submitted as journal papers [34] [30]. In addition, the gained knowledge can be exploited in form of best practice guides.

The software toolkit for modal decompositions developed by RWTH was further enhanced with a modular reader interface and applied to the Alya use-case C2U1 to investigate the hydrodynamic instability in a swirl-stabilized combustor. The software toolkit itself and practical experience with its application to large-scale simulation data will be basis for a hands-on workshop on “Data Analysis using Modal Decompositions”. Additional documentation material and user guides for the execution of a DMD analysis are planned to be compiled.

For TPLS, UEDIN investigated in-situ data analysis but found it to be an inefficient addition. Instead, the associated data analysis of TPLS has now been updated to reflect the improved data management which was, in turn, introduce to prepare for exascale platforms.

5 Task 4.4: Data management

5.1 Introduction

The platform combines the topics data transfer, data management, data dispatching and data scheduling. Therefore, the platform merges the two work package efforts “Data dispatching through data transfer (WP3.6)” and “Data Management (WP4.4)” since they can’t be separated.

A typical data round trip for an exascale simulation workflow is shown in Figure 17 - pointing out several challenges. The round trip describes the overall process, that we are trying to adapt and understand with the help of a first proof of concepts.

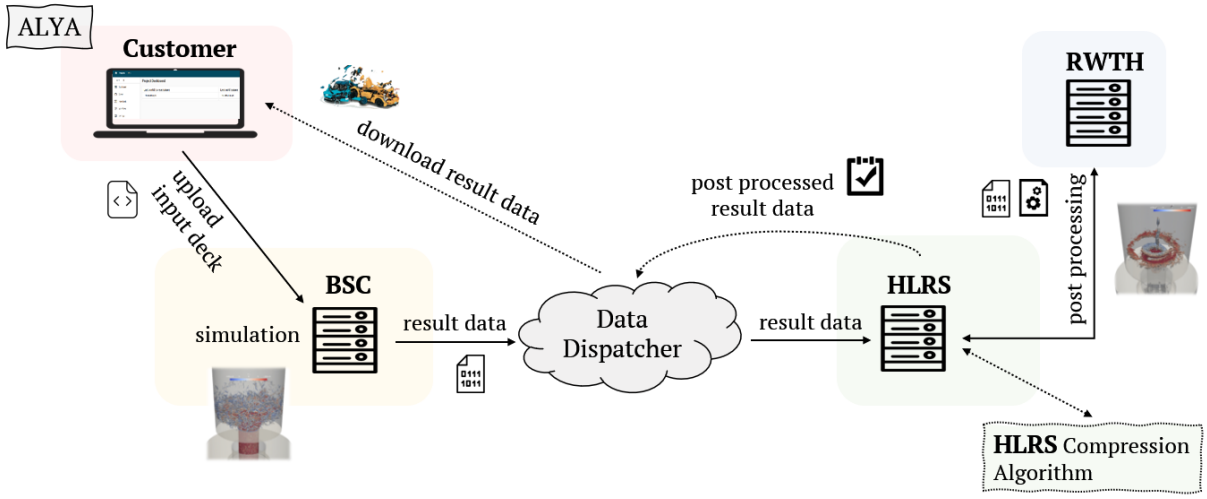


Figure 17: Data Roundtrip.

The goal is to complete a data transfer round trip with one of the EXCELLERAT use cases, that covers all aspects of an Engineering workflow - data simulation, data transfer, data analysis, data post processing and the transfer back to the customer. In many cases only the code itself was optimized, but it was often forgotten that there is a certain lifecycle around it that is especially important for the end user.

5.2 Challenges regarding Exascale

Companies face various problems while dealing with HPC calculations, HPC in general or even the access to HPC resources. Small companies in particular can't meet certain requirements because their existing internet connection does not meet the previous requirements of the necessary setup for HPC connection.

Input data and application code have to be sent to an appropriate computing cluster, that can handle the exascale simulation. The simulation results then usually consist of a very high resolution and thus large amounts of data. As transfer bandwidth and storage are restricted, it has to be considered carefully which data is sent back to the client. Altogether, data transfer needs to be efficient and secure.

We attack these challenges in several tasks of EXCELLERAT, which is clearly described in the following section.

5.3 Solution

Through the Data Exchange & Workflow Portal, which is developed within the Tasks 3.6 and Task 4.4 by SSC, these challenges can be eliminated. The platform is not only used for data processing, but will also enable a safe and traceable, bidirectional, online data transfer between the data generators and all six high-performance computing centres represented in the EXELLERAT project. This data transfer will be highly automated to avoid duplication of the transferred content. This approach will lead to a data reduction, which can ultimately save time and costs. The portal provides all relevant HPC processes for the end users, such as uploading input decks, scheduling workflows, or running HPC jobs.

The added value of the workflow portal in relation to exascale data are the topics such as data reduction, volume reduction and data compression of input and output data. In concrete terms, this means that the data is getting small and manageable for the data transport.

In Task 3.6 and Task 4.3, FRAUNHOFER and RWTH develop data analysis and reduction techniques, that extract as much relevant information as possible from the high-dimensional simulation data into a compact representation, such as physical or data-driven features. The algorithms will be executed on the same architecture as the main simulation. Besides providing better physical insights, this will reduce the amount of data that needs to be transferred back to the user.

Future efforts will be needed on data management activities to organize simulation runs which is, however, beyond of the scope of EXCELLERAT. Keeping detailed track of simulations that have already been executed including all in- and output data could simplify the analysis process and prevent redundant calculations, thus saving computing efforts. A smart simulation management can also include an on-demand selection of computing resources based on hardware requirements and resource availability.

5.4 Introduction of the platform interface

In the following section the progress of the platform so far is shown and explained graphically.

After a successful login, the user starts on a dashboard page. To be able to use the corresponding HPC resources, a connection to the cluster, on which the calculation is to be performed, is required. Currently, HLRS is available with the HPE Apollo (Hawk) and the NEC cluster (Vulcan). To connect the platform with the machine user, the SSH access data of the corresponding cluster must be entered once.

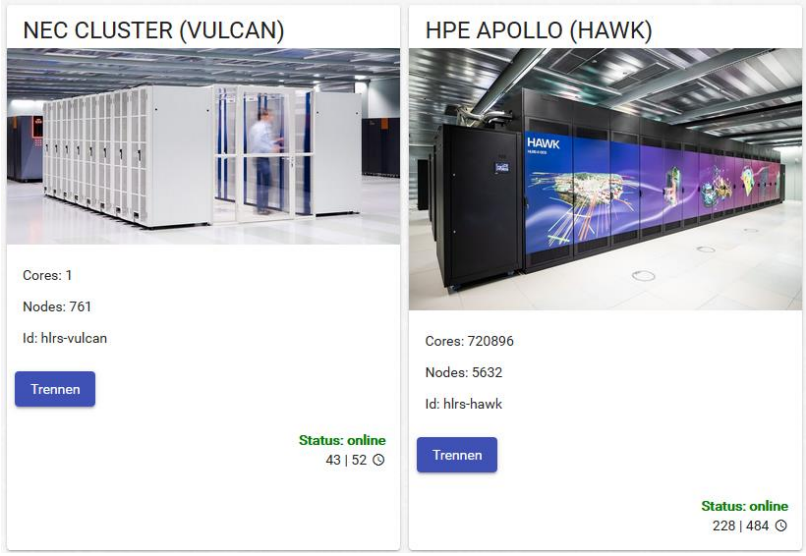


Figure 18: Connection to an HPC cluster.

The basic structure of the platform consists of projects corresponding to the workspaces on the clusters. This means, that when creating a project, a workspace with the same name is created on the cluster. When a new project is initially created, a name must be assigned. In addition, you must specify how long the retention period (1-30 days with a possible extension of three times, which corresponds to a total of 120 days) should be. A description is optional.

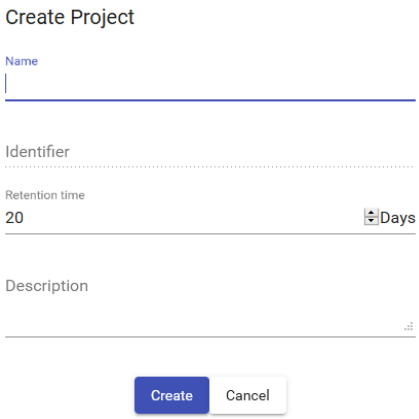


Figure 19: Project creation.

The project contains a dashboard, an input deck, workflows, and project settings. The dashboard shows the last executed workflows and the latest notifications. To prepare the simulation, data is created locally and can be uploaded to the corresponding cluster through the input deck menu. After the appropriate files have been selected, the user is asked on which machine these files should be uploaded. The background is that in the future, further HPCs with several machines will be connected to the platform and the goal is, that the platform will decide, where which code is best stored for calculation. The progress of uploading the local files is displayed to the user in the interface. During the upload, the system checks whether there are already identical file pieces, that are already in use and do not need to be uploaded again.

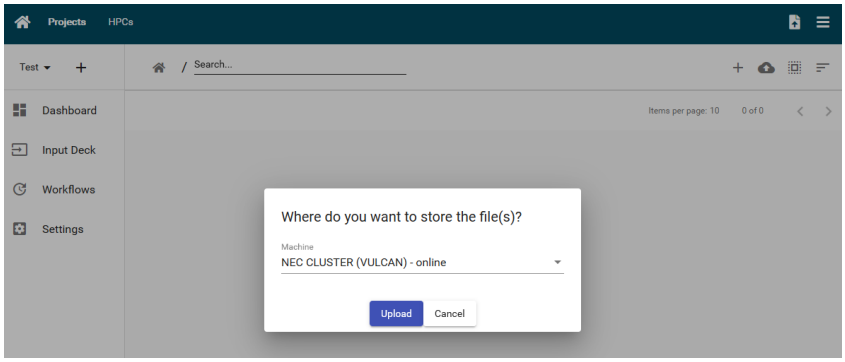


Figure 20: Uploading an input deck.

All uploaded files end up in the editor's workspace. For the platform, a specific folder structure is created on the HPC cluster so that all files can be found again. However, all modifications, that take place outside the platform are not monitored, since no direct accesses are integrated into the cluster. The following folder structure is used: The "input deck" contains all input data, that have been uploaded and "runs" contains all executions with the result data.

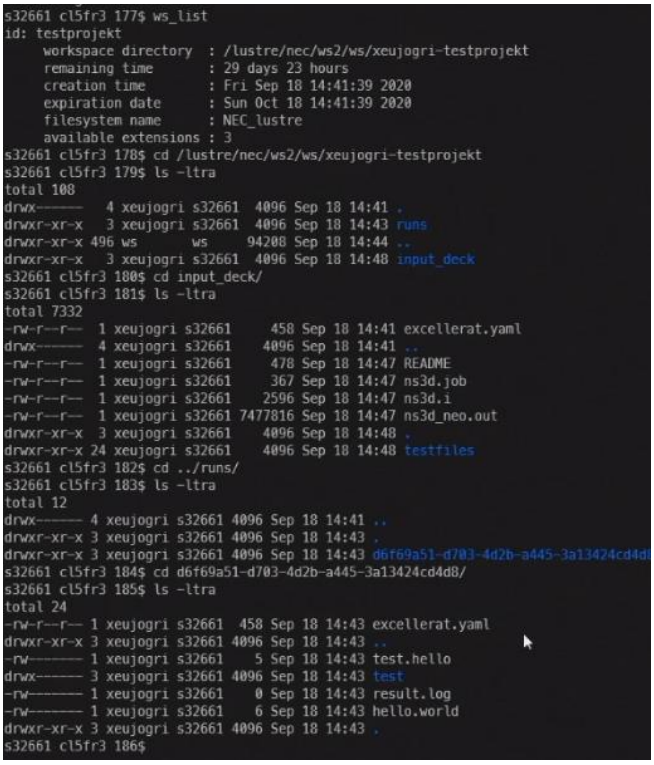


Figure 21: Workspace structure on HPC filesystem.

So far you can specify an "excellerat.yaml" file for each project. This control file describes the workflows and how the simulation should look like and what should be done with it in the workspace. You can also specify scripts here that may run in pre- and post-processing. These could be included in the YAML file as batch scripts or uploaded at the beginning and called in YAML.

```yaml
steps:
- name: sleep
  uses: actions/run
  runsOn: hlrs-vulcan
  script:
    - "for i in {1..10}; do echo 'console output' $i; done"
    - "sleep 5s"
- name: sleep
  uses: actions/run
  runsOn: hlrs-vulcan
  script:
    - "mkdir -p test/1/2/3"
    - "touch result.log"
    - "touch test/result_test.log"
    - "touch test/1/1.log"
    - "touch test/1/2/2.log"
    - "echo hello > hello.world"
    - "echo test > test.hello"
    - "sleep 5s"
```

Figure 22: Example of an excellerat.yaml file

After creating a new workflow, the uploaded excellerat.yaml is automatically loaded into this workflow. In the background, the file is loaded from the cluster workspace to be displayed in the browser. Additionally, the workflow can be changed manually. There is a validation of the commands and an auto-completion. Around the control file the input data is added to run the simulation. The corresponding workflow can be scheduled and started after pressing "Schedule". The platform component in the corresponding cluster executes the command for e.g., "qsub" and prepares the run script ("run.sh") from the input files and passes it to the simulation. Starting the run is started here.

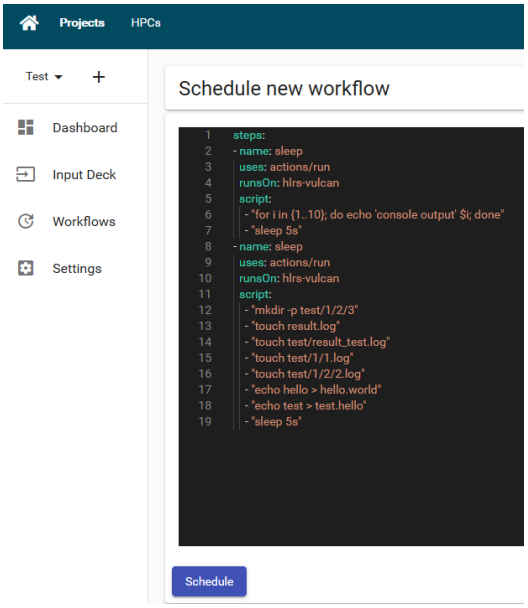


Figure 23: Workflow scheduling.

After the run has been successfully scheduled, each step is processed in the background, executed automatically and the user receives a browser pop-up notification at the end that the run has been successfully completed. In this example, a few files and a folder structure were created in the workspace, which the user can now download.

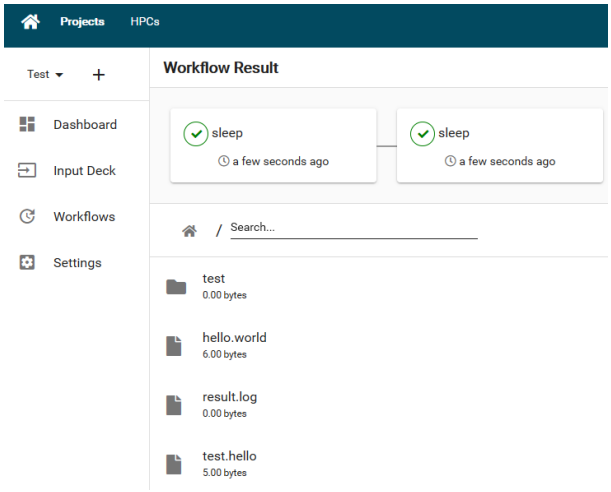


Figure 24: Workflow overview.

5.5 Development Progress

Since the last Deliverable, there has been a major step forward in the back-end development of the platform's intelligent data handling. The goal is the data reduction of input data to make the data small and manageable for data transport. This data reduction part now takes place in the background of the platform and has been implemented. When uploading, a hash tree is first calculated from the whole file. This means that the file consists of many individual megabyte blocks, each of which has a unique hash. This hash tree can be used to determine which parts have been uploaded before. In the future, only changes have to be uploaded again. The next step of the reduction is to determine the media types of a file to see how much they can be compressed before they are uploaded. The last step, which is not yet fully implemented, is to compress the files on the way back.

Furthermore, many small features have been added. It is now possible to connect more than one HPC to the platform. Users can connect directly to the HPE Apollo (Hawk) and the NEC cluster (Vulcan) of the HLRS with their user information via the platform. In addition, the user is shown how many jobs are currently running on the respective system.

Additionally, the last executed workflows are now displayed to the user on the Project Dashboard Overview. Here the user can see at any time in which state the executed job is.

Parallel to the further development of the platform, SSC is currently in the process of testing the platform and its feasibility with selected pilot partners and incorporating their user feedback into the platform. The knowledge gained from this can be incorporated into the development of the platform on the one hand, and on the other hand these results also flow back into the EXCELLERAT project.

5.6 TPLS I/O for exascale platform

In addition to that, UEDIN has prepared TPLS I/O for exascale platforms. They have investigated efficient strategies for pre- and post-processing as well as I/O, storage, accessibility, and efficient metadata management for TPLS. Following closely the definition of

Task 4.4, this has particularly included parallel I/O strategies, e.g. MPI-IO and parallel file formats, like HDF5 and NetCDF, the development of data reduction strategies and, to a lesser extent, the exploitation of novel memory and I/O hierarchies.

It is worth noting, that UEDIN's work as part of Task 4.4 does not fall within the remit of the Task's associated Working Group on Workflows.

TPLS I/O has now been modernized and is ready for exascale usage. This was done by replacing serial I/O with parallel I/O routines, and by fully enabling its restart functionality. Three data reduction strategies were introduced Firstly, by replacing the ASCII data formats with binary HDF5 files, and secondly by enabling the end user to specify which data to read/write rather than reading/writing unwanted data. And finally, by removing cell coordinates from data files, as these can be reconstituted by the location of data values within the HDF5 array along with mesh structure metadata contained in associated, small human-readable XDMF metadata files. Note that the HDF5 files were to be created using NetCDF; however, it was found that, since TPLS predominately employs PETSc, the PETSc HDF5 I/O routines proved to be far more efficient in terms of memory usage, usability, source code maintenance, and overall performance. Efficient metadata management was introduced via the aforementioned XDMF files, where each small human-readable file describes its associated binary HDF5 file. Novel memory and I/O hierarchies will be catered for by the PETSc I/O library, as and when PETSc releases new versions for new hardware.

5.7 Summary of Data Management Activities

The main focus of the work in task 4.4 and task 3.6 has been to put on the development of the platform and to complete the data roundtrip for exascale engineering simulations. The most important outcomes are listed below:

- Providing the most relevant HPC processes to the users (upload Input Deck, run simulation, download results)
- Secure, traceable, automated transfer between data generators and HPCs
- Various cosmetic changes in the platform's user interface to improve usability
- Within the WP-transversal data transfer and data management working group, the data transfer task (task 3.6) and the data management task (4.4), we could contribute to the EXCELLERAT service portfolio. For example, the platform was integrated into the service platform, developed in WP5.
- Time and cost savings due to a high degree of automation that streamlines the process chain
- The platform can be further optimized and opened up for more HPC centres, gaining one tool to address all centres

Knowledge gained during the development of the data workflow will be exploited in best practices guidelines, training activities as well as consultancy services for support and management of the large amount of data produced and used in engineering applications.

6 Task 4.5: Usability

The general goal of the usability task is to provide workflow and best practices for an engineering simulation's entire life cycle, from pre-processing, including modelling and meshing, execution of simulations to post-processing. In order to derive efficient workflows, this task gathers the expertise and methods developed within EXCELLERAT necessary to run large-scale engineering simulations.

In this reporting period, the usability task has taken an observatory role, monitoring the developments in the core codes and gathering experiences from use-case owners after their initial set of runs. A common topic across several core codes in this reporting period is the use of emerging technologies and accelerators. Up until recently, most if not all, accelerator related efforts within the centre have focused on Nvidia technology. Using OpenACC or CUDA core codes have gained significant experience in how to exploit these technologies. However, in light of the recent EuroHPC announcements [36] and announced DoE exascale systems [37], support and knowledge about OpenMP 5 (offloading) or HIP [38] might be as crucial for developers of engineering simulation codes. With an expected ramp-up in the use-case activities, with larger and more data-intensive runs in the final phase of the project, Task 4.5 will intensify its effort to gather data and post-processing experiences from code and use-case owners. Based on the gathered information, guidelines and best practices will be devised and reported in a separate deliverable D4.5 *Best Practices and Tools for Visualization, Data Management and Analytics of Engineering Simulations at Exascale*.

7 Conclusion

To conclude, the development of enhanced services to support exascale engineering simulation workflows is progressing as planned within work package 4. The significant achievement in this reporting period has been to formulate prototypes of the enhanced services outlined in deliverable D4.2. Co-design activities are established with a strong connection to work performed in work package 3. Scalable in-situ visualization workflows for both interactive and non-interactive analysis has been developed and tested with several of EXCELLERAT's core-codes. Similarly, in-situ data analytics and uncertainty quantification frameworks are developed and validated with different codes and problems. HPC specific data management techniques have been developed and tested with a complete data roundtrip from uploading of an input deck to downloading of post-processed results. These services have been designed and implemented based on use-case's needs. Either using derived model problems or scale down formulations of the full use-cases. The plans for the remainder of the project are to continue developing the enhanced services, advanced past, current prototypes and focus on the integration of the development into the full use-cases in EXCELLERAT.

8 References

[1] POP, “Performance Optimisation and Productivity -- A Centre of Excellence in HPC,” 2020. [Online]. Available: https://pop-coe.eu/.

[2] CompBioMed, “A Centre of Excellence in Computational Biomedicine,” 2020. [Online]. Available: https://www.compbiomed.eu/.

[3] ETP4HPC, “ETP4HPC - European Technology Platform for High Performance Computing,” [Online]. Available: https://www.etp4hpc.eu/.

[4] GENCI, “TGCC,” [Online]. Available: https://www.genci.fr/fr/content/calculateurs-et-centres-de-calcul.

[5] EPCC, “FULHAME,” [Online]. Available: https://www.epcc.ed.ac.uk/facilities/other-facilities/fulhame.

[6] E4, “ARMIDA,” [Online]. Available: https://www.e4company.com/en/2020/06/energy-efficiency-in-the-hpc/.

[7] IDRIS, “JEANZAY,” [Online]. Available: http://www.idris.fr/eng/jean-zay/].

[8] CINECA, “Marconi100,” [Online]. Available: https://wiki.u-gov.it/confluence/pages/viewpage.action?pageId=336727645.

[9] BSC, “POWER CTE,” [Online]. Available: https://www.bsc.es/support/POWER_CTE-ug.pdf.

[10] CINECA, “Galileo,” [Online]. Available: https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.3%3A+GALILEO+UserGuide.

[11] CINECA, “Marconi,” [Online]. Available: https://wiki.u-gov.it/confluence/display/SCAIUS/UG3.1%3A+MARCONI+UserGuide.

[12] HLRS, “Vulcan,” [Online]. Available: https://kb.hlrs.de/platforms/index.php/Aurora_Tsubasa.

[13] EPCC, “Maxwell,” [Online]. Available: https://www.epcc.ed.ac.uk/projects-portfolio/fhpca-supercomputer-maxwell.

[14] ARM, “GENCI Journey of optimizing applications on arm platforms,” 2020. [Online]. Available: https://community.arm.com/developer/tools-software/hpc/b/hpc-blog/posts/genci-journey-of-optimizing-applications-on-arm-platforms) .

[15] P. Fischer and K. Heisey, “NEKBONE: The Thermal Hydraulics mini-application,” 2013. [Online]. Available: https://github.com/Nek5000/Nekbone.

[16] N. Brown, “Exploring the acceleration of Nekbone on reconfigurable architectures,” in *Sixth International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC),* 2020.

[17] Nekbone, “UserGuide,” [Online]. Available: https://github.com/Nek5000/Nekbone/blob/master/USERGUIDE.pdf.

[18] N. Jansson, “Spectral Element Simulations on the NEC SX-Aurora TSUBASA,” in *The International Conference on High Performance Computing in Asia-Pacific Region*, 2021.

[19] “Vistle,” HLRS, [Online]. Available: https://vistle.io/.

[20] “VisIt,” Lawrence Livermore National Laboratory, [Online]. Available: https://wci.llnl.gov/simulation/computer-codes/visit/.

[21] “SENSEI · Scalable in situ analysis and visualization,” Kitware, [Online]. Available: https://sensei-insitu.org/.

[22] “VTK,” Kitware, [Online]. Available: https://vtk.org/.

[23] “ParaView,” Kitware, [Online]. Available: https://www.paraview.org/.

[24] “Python,” [Online]. Available: https://www.python.org/.

[25] “MareNostrum 3,” Barcelona Supercomputing Center, [Online]. Available: https://www.bsc.es/marenostrum/marenostrum/mn3.

[26] “OpenGL,” [Online]. Available: https://www.opengl.org//.

[27] J. Kress, M. Larsen, J. Choi, M. Kim, M. Wolf, N. Podhorszki and S. Klasky, “Comparing the efficiency of in situ visualization paradigms at scale,” in *International Conference on High Performance Computing*, 2019.

[28] “Jupyter,” [Online]. Available: https://jupyter.org.

[29] “Scikit-learn,” [Online]. Available: https://scikit-learn.org.

[30] S. Rezaeiravesh, R. Vinuesa and P. Schlatter, “UQit: A Python package for uncertainty quantification (UQ) in computational fluid dynamics (CFD),” *submitted to JOSS,* 2020.

[31] “Mpi4py,” [Online]. Available: https://mpi4py.readthedocs.io.

[32] “Elemental,” [Online]. Available: https://github.com/elemental/Elemental.

[33] “ParaView Catalyst,” Kitware, [Online]. Available: https://www.paraview.org/in-situ/.

[34] S. Rezaeiravesh, R. Vinuesa and P. Schlatter, “An uncertainty-quantification framework for assessing accuracy, sensitivity, and robustness in computational fluid dynamics.,” *arXiv:2007.07071,* 2020.

[35] “D2.2: Report on Reference Applications Outcomes”.

[36] CSC, “LUMI,” 2020. [Online]. Available: https://www.lumi-supercomputer.eu/lumi-one-of-the-worlds-mightiest-supercomputers/.

[37] Oak Ridge National Laboratory, “Frontier,” 2019. [Online]. Available: https://www.ornl.gov/news/us-department-energy-and-cray-deliver-record-setting-frontier-supercomputer-ornl.

[38] AMD, “ROCm Documentation,” 2020. [Online]. Available: https://rocmdocs.amd.com/en/latest/Programming_Guides/HIP-GUIDE.html.

[39] G. J. Pringle and e. al., “D2.2 Report on Deployment of Deep Track Tools and Services to Improve Efficiency of Research and Facilitating Access to CoE Capabilities,” CompBioMed, 2019.

[40] The Performance Optimisation and Productivity Centre of Excellence in Computing Applications, [Online]. Available: https://pop-coe.eu/.

9 Appendix

9.1 Rough Guide to Preparing Software for Exascale

This Appendix Section is to be used as a Crib Sheet for improving the efficiency of software and will be published as a best practice guideline on the EXCELLERAT service portal.

The test has been adapted from Appendix A of the CoE CompBioMed deliverable [39].

The term exascale is used to describe HPC hardware capable of at least one exaFLOPS, or 10^{18} FLoating point OPeration per Second. It is envisioned that such machines will have many multi-core processors, and that the available memory per core will be far inferior to those on current HPC platforms. For instance, this was seen when attempting to port MPI codes to IBM Blue Gene machines, or the Intel Xeon Phi family, where the amount of memory per core is prohibitively small for many codes parallelised using MPI only. As such, the common practice of running one MPI task per physical core may no longer be possible for the majority of codes in the future.

The solution for getting codes ready for exascale platforms requires both software and hardware related strategies. The former, the subject of this note, is described below. The latter, beyond the scope of this note, is achieved via Co-Design, where hardware vendors and end users work together to ensure future platforms are not built to achieve exascale performance at the expense of usability.

Application codes rarely perform and scale well when first parallelised: each doubling of scale typically exposes a new issue. Ensuring the application will scale on HPC systems - both today and in on the exascale systems of the future - requires stepwise increasing of scale and validation of correctness, debugging, performance analysis and tuning. Then, repeat for each significant code extension/optimisation. Through performance analysis, programmers can locate so-called “hot spots”, i.e. code which takes the most time, as this code should then be targeted for improvement.

9.1.1 Software preparations

Given the memory per core will most likely be substantially reduced when compared to today’s HPC platforms, the practice of assigning one MPI task per physical core will have to be substituted by using every 2nd or 4th core for each MPI task. This is known as under-populating nodes. At first glance, this appears to suggest that we cannot fully exploit the hardware, as we simply avoid using 50% or even 75% of the cores; however, these spare cores can be employed via mix-mode codes, where each MPI task runs threaded routines/loops to run on the remaining cores.

Essentially, authors must expose as many levels of parallelism as possible within their code. Coding this can involve ensemble runs to coupling multiscale codes, multiprocessing (with inter-process communication) to multithreading, vector processing to accelerator-specific

commands. This process can slow the code down on present day platforms but will future-proof the code.

For instance, there are sets of serial algorithms, so-called Optimal Serial Algorithms, which are often difficult or simply impossible to parallelise, as these algorithms employ data from the previous steps or even the current step to make improvements at the current step. Such dependencies can prevent concurrent execution of threads in the program, for instance.

The inelegant yet empowering solution is to replace the optimal serial algorithm with a sub-optimal serial algorithm which is, however, parallelisable. Whilst the serial performance may be worse, the parallel performance will soon outperform the serial version as the number of cores increases.

9.1.2 Improve serial code

Before considering how the code is parallelised, the first step is to consider the serial sections of the code.

- Remove excess memory use in serial code.
- When using C++, find good balance of OOP and functional programming, as an intensive use of OOP might introduce an unnecessary layer of complexity of the scientific code.
- Ensure proper use standard libraries

9.1.3 Introduce vector processing

- Use appropriate compiler options
- Write ordered loops or leave this to compilers?
- Innermost loop must have independent iterations
- Loop length is either larger of multiple of vector length
- It is possible to set this at compiler time but not "probe and populate"
- No function calls, except maths libraries
    - functions can be vectorised using OpenMP “declare simd” feature
- No complex control flow
- Determinable trip count (i.e. no while)
    - the trip count must be known before entering the function at runtime
- Data access should be vector aligned, i.e., start at vector boundaries, and preferably continuous
- For more advice on vector processing: http://www.archer.ac.uk/training/course-material/2017/11/sgl-node-ox/L04-vectorisation.pdf
- Be aware of the ISA (Instruction-Set Architecture), such as SSE (Steaming SIMD Extensions, AVX (Advanced Vector Extensions), etc.

- it determines the vector length
- may target vectorised FMA instructions
- Do loop padding manually to get rid of peel/remainder loops
- Concerning vectorisation, we check compiler output or assembler code to see what was vectorised
- Use inline hints for functions or routines to help out the compiler to inline
- Remember that YOU know your application better than the compiler does.
- It all depends on how the data is aligned in RAM

- Hints with pragmas might be useful, also
- Force data alignment with compiler instructions (usually done automatically by the compiler)

9.1.4 Improve MPI code

- MPI messages should be grouped to avoid multiple smaller messages
  - e.g. use derived data types to avoid double buffering
- Avoid any storage or computation of *O*(nranks)
- Avoid all-to-all communication
  - e.g. if(rank==0)then do work over all other ranks
- Remove unnecessary MPI_BARRIERs
- Do not over schedule cores when using threaded maths libraries
  - typically control using OMP_NUM_THREADS even when libs do not use OpenMP
- Use nonblocking collective communications.
  - overlap computation and communications where possible
- Remove unnecessary communication synchronisation
  - use MPI_TEST rather than MPI_WAIT
  - avoid MPI_Probe
    - it most likely forces internal buffering to report the size of the pending message
  - Avoid ordered halo swapping,
    - e.g. do not delay y-direction sends until x-direction receives have completed.
    - however, huge network bursts are also not ideal

- sometimes, ordered sends allow ordered receives.
- and ordered sends might allow to take advantage of the network topology
    - e.g. one can completely load the network with x-direction halo swaps and so on
- Ensure load is balanced
    - Avoided the receive-before-send scenario
        - one-sided communications can alleviate this
- Be aware that not all MPI libraries are equal
    - e.g., there are many ways to implement collective communications.
- Respect the fact that the MPI standard prohibits concurrent read accesses on the same buffer (even though there is no race condition)
    - It may reduce the efficiency (or cause bugs)
- Tag the source
- Be aware: “blocking” has an alternative meaning in the MPI standard.
- This can easily lead to serialisation of huge chunks of the program.
- Interleave/overlap communication with computation where possible

9.1.5 Improve MPI parallelism

- Give each MPI task multiple sub-domains
    - a subdomain is a distinct region of the computational domain and a result of the domain decomposition algorithm.
    - this allows light weight parallelism on a socket, keeps cache logically together, etc.
- Enable multiple tiles per task.
    - this might make tiles fit into cache but will spend time swapping boundary information with yourself
- Use MPI Communicators, to map the communication to the target HPC topology
    - Collective operations are possible on a subset of processes.
    - Explicit communicators are very useful to leverage MPI Shared Memory
- One-sided communication (or Remote Memory Access (RMA)), can be faster than the message passing model
    - May be beneficial when the load is hard to balance, since delays in the receiving process are not necessarily propagating to the sender

9.1.6 Introduce OpenMP for threads on cores and OpenACC for GPUs

A code which uses both MPI and OpenMP, or a code that uses both MPI and OpenACC, is referred to as a mixed-mode code. This is done for two reasons: reduce memory footprint or/and speed up application.

Not all MPI codes benefit from becoming mixed-mode codes. The benefits are as follows.

- Hybrid applications have a reduced memory footprint (the shared memory model allows threads to avoid halo regions or ghost cells)
- Eases load balance issue (usually the complexity of (adaptive) load balance growths with the number of subdomains)
- Load balancing in threads is much easier
    - thread-pool model, or
    - tasks
- For applications which are MPI-bound due to load imbalance (long barriers in MPI_Wait or MPI_Receive/Send), it might be advisable to reduce the number of processes and increase the threads, while using OpenMP's built in load balancing features

Whilst the drawbacks are as follows:

- In case of MPI_THREAD_MULTIPLE, the application might lose portability
    - forked threads are allowed to call any MPI routines
- Shared memory applications have their own problems
    - e.g. false sharing, where a cache line is voided repeatedly
        - this is naturally avoided by MPI processes
- NUMA effects, e.g. where the data is placed in memory
    - this can be resolved by careful task mapping.

Maybe also better to use OpenMP 4.5 target directives than OpenACC.

9.1.7 Improve OpenMP parallelism

An excellent Best Practice Guide for writing mixed-mode programs, i.e. MPI+OpenMP, can be found via the Intertwine project pages: https://www.intertwineproject.eu/mpi-plus-openmp-threads-resource-pack

- Investigate OpenMP tasks
- Try different schedules and/or tasks
- Avoid over-scheduling threads when calling threaded maths libraries.
- Minimise sequential code
- Replicating computation rarely works

- Ensure load balance over threads.
    - Use different loop schedules or tasks
    - May includes balancing communication in one thread with calculations in the rest
- Avoid MPI data types as packing data is done on one thread: better to pack data in parallel using threads, as MPI should not need to double pack when data is contiguous
- Take care with NUMA effects my considering mapping, i.e. task placement
    - e.g. run at least one MPI process per NUMA node
- Take care with process and thread binding: threads should run on the same socket as their parent MPI process.
- Minimise the number of OpenMP barriers
- Use OpenMP directives to force SIMD operations
    - OpenMP allows explicit vectorisation of functions called from vectorised loops

9.1.8 General programming tips

- Be aware of the Load-Hit-Store problem (it does exist on multiple levels)
    - prevents caching by the compiler and causes pipeline stalls.
    - e.g., appears in MPI-IO (sometimes referred to as Read-Modify-Write effect)
- IO can dominate
    - consider MPI-IO or, better still, HDF5

9.1.9 Code Longevity

Whilst this section includes good practice for software engineers in general, the following points are key when preparing for exascale systems, primarily because large popular codes outlive the programmers who, in turn, typically outlive the HPC platforms for which the software was written.

- follow a strict coding style guide
- use readable variables
- use internal documentation
- keep routines to less than one page
- copyright statements for every module/subroutine/function
    - key for IP monitoring

9.2 Using Vistle's in situ capabilities

This Appendix Section provides a brief user guide on how to utilise Vistle's in situ capabilities developed within EXCELLERAT.

9.2.1 For LibSim instrumented simulations

The LibSim instrumentation on the simulation side is done by linking to a small static LibSim library and provide call-back functions for it to retrieve the data from the simulation. When the simulation starts, this static library creates a TCP socket on rank zero and dumps a file with the connection information. As soon as the simulation receives a valid connection request, it links dynamically to a library called libsimV2runtime_*suffix*.*ending* with *suffix* "par" if build with MPI and else "ser" and with *ending* "so" on Unix and "dll" on Windows systems. Vistle therefore creates such a library (further referred to as "the adapter") implementing the same interface as VisIt but with Vistle's own implementation (CMake variable LIBSIM_PARALLEL can be used to tell Vistle weather to build the parallel or serial LibSim library).

Before the start, some environment-variables have to be set so that the simulation finds the Vistle libraries and to make sure Vistle uses the same MPI-environment as the simulation. Table 2 gives an overview of these variables and when and for which environment to set them. For the simulation variables it has to be ensured that they are passed to all MPI ranks.

In multi-process mode Vistle can be connected by starting Vistle and launching the LibSim-controller module, where the connection file created by the simulation must be opened. Figure 25 illustrates the data exchange in this case. On rank zero the module will connect to the connection-socket and send a connection request to the simulation. If the request is valid the simulation closes this socket and links to the adapter. The adapter is initialized with the arguments of the connection request message. The adapter then connects to the modules control-socket which is used to send meta data about the simulation to the module and to steer the simulation via commands. With this meta data the module creates output ports for the simulation provided data and buttons for the simulation's commands. Data objects are transformed and copied to SHM in the adapter and then passed to the Vistle pipeline if the corresponding output port is connected, the module is executing and the simulation timestep matches the transmission frequency.

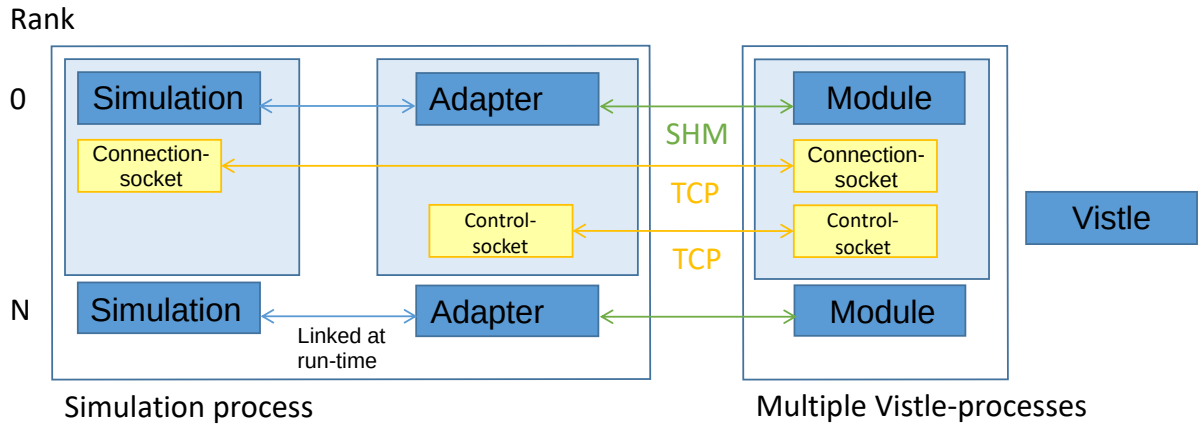


Figure 25: Schematical data exchange between a LibSim-instrumented simulation and Vistle in multi-process mode

In single-process mode Vistle has to be started in the simulation process. Therefore, the Vistle hub is used to send the connection request to the simulation to trigger the linkage to the adapter (pass the path to the connection file with the --libsim command line argument to Vistle). The adapter then launches Vistle in a separate thread and the module can be started. For this to work the simulation must use MPI_Init_thread with MPI_THREAD_MULTIPLE since Vistle and its modules all use dedicated threads and may make MPI calls. Figure 26 shows how data is exchanged with this architecture.

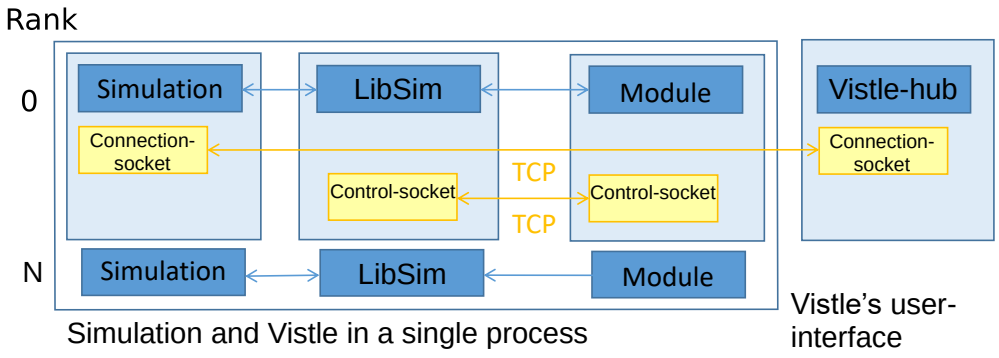


Figure 26: Schematical data exchange between a LibSim-instrumented simulation and Vistle in single-process mode

In both modes the connection via the control-socket is exclusive to rank zero. The messages are then broadcasted via MPI to the other ranks so that the commands can be executed parallelly. The commands are declared by the simulation via the LibSim interface and can be triggered through buttons in the controller module. As shown in Figure 2 the module displays the data provided by the simulation with its output ports. To retrieve the data from the simulation the corresponding output ports must be connected and the module must be executing (it will stay in executing state until cancelled). This way data selection can be adapted and the post-processing can be started/stopped at any point during runtime to avoid overhead of transforming data when it is not needed.

9.2.2 For SENSEI instrumented simulations

To enable the Vistle post-processing-backend our SENSEI version has to be built with the ENABLE_VISTLE flag. The Vistle backend is then compiled and linked with Vistle's libraries, therefore using a single- or multi-process Vistle a compile time decision for now (Adding two SENSEI backends for Vistle might be possible as well). Like for LibSim some environment variables, as shown in Table 2, have to be set before the connection process. The whole connection and communication process between the SENSEI analysis adaptor and Vistle's SenseiController-module is implemented via SHM communication. The backend also transforms SENSEI's VTK objects to Vistle's format. In case of a multi-process Vistle they are copied to SHM while they are passed as pointers in the single process variant. In single-process mode the backend will directly start Vistle without a user interface (UI). Again MPI_Init_thread with MPI_THREAD_MULTIPLE is required for this to work. The UI can be attached by simply connecting a Vistle UI (vistle_gui or vistle_tui) to the simulation's Vistle instance. In multi-process-mode Vistle can be started regularly. The SenseiController-module has to be started and a connection file outputted by the backend must be loaded. This module supports only basic steering like "run" and "pause", otherwise the functionality is similar to the LibSim module.

Environment variable	Value	Single-process	Multi-process	Vistle-Shell	LibSim-Shell	SENSEI-Shell
MPISIZE	MPI-size of the simulation		X	X		
MPIHOSTS	MPI-hosts of the simulation (or MPIHOSTFILE)		X	X		
MPIHOSTFILE	Host-file of the simulation (or MPIHOSTS)		X	X		
VISTLE_KEY	Same key of four digits	X		X	X	X
VISTLE_ROOT	Path to Vistle's build directory	X			X	X
LD_LIBRARY_PATH	Add path to Vistle's lib directory	X	X		X	

Table 2: Environment variables needed to run simulations in-situ with Vistle