H2020-INFRAEDI-2018-2020

The European Centre of Excellence for Engineering Applications

Project Number: 823691

D4.7

Final 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
	Jens Gerle	SSC
	Ricard Borrell	BSC
	Simon Loosen	RWTH
	Michael Wagner	DLR
	Mirco Valentini	CINECA
Approved by	Executive Centre Management	
Reviewer	Claudio Arlandini	CINECA
Reviewer	Lorenzo Zanon	HLRS
Dissemination Level	PU	

Date	Author	Comments	Version	Status
2022-04-29	N. Jansson et al.	Initial contributions	V0.1	Draft
2022-05-06	N. Jansson et al.	Draft ready for QA	V0.2	Draft
2022-05-23	N. Jansson et al.	Addressing review comments	V0.3	Draft
2022-05-29	N. Jansson et al.	Final updates	V1.0	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*
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*
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*
SPLISS	*Sparse Linear Systems Solver library*
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 developments and significant achievements in work package 4 throughout 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 during the project 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. In the final phases of the project, developed services have advanced past the prototype stage and have been integrated and validated within 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 throughout 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 6 for inclusion into the EXCELLERAT's service portfolio.

The deliverable is structured as a progress report. Each task in work package 4 gives details of the work done, and of the progress towards realising the envisioned services outlined in deliverable D4.2 *Report on the Service Portfolio.*

2 Task 4.1: Co-design

The Co-Design Service continued to support our Reference Applications, and the service offering within the portal has been improved dramatically.

The EXCELLERAT Service Portal contains the Co-Design Service offer[1] within the Consultancy Topics. This offer has been re-worded to encourage both SMEs and less experienced potential clients via more welcoming text and imagery. Moreover, a joint effort with work package 5 to reword and refactor the sidebars of this co-design service offer has led to improvements throughout the Service Portal.

As previously declared in D4.6 *Report on Enhanced Services Progress*, EXCELLERAT's Co-Design Service follows an *indirect* co-design paradigm, wherein clients (including Core Partners, of course) gain access to the early release of state-of-the-art hardware, which is available typically due to a close working relationship between the vendor and the Core Partner. This avoids a *direct* co-design paradigm, given vendors typically will not alter their hardware to benefit a small set of applications. This indirect co-design paradigm permits our Co-Design Service to exploit trends in software and hardware and match them to the code design issues of our clients' applications. To date, all clients have been Core Partners along with their Reference Applications running our use cases.

We have maintained our 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,

[1] https://services.excellerat.eu/viewservice/1

- 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; 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].

The Co-Design Working Group agreed to the following methodology:

- Firstly, target Reference Applications are chosen.
- Full applications may be reduced to smaller *proxy* applications,
    - where each proxy is a small bundle of highly portable source code, with example input and output files,
        - these used to be referred to as *mini-apps*
    - which retain the computational characteristics of the full simulation, e.g., computational kernels are retained
        - key data movements epochs, such as I/O, might also be retained.
    - This can circumvent irrelevant porting issues that can arise when addressing novel architectures.
- The full code, or its proxy, is then ported to the existing hardware or emulators of future hardware,
    - where associated libraries may also need to be installed.
- The target simulation is also provided,
    - and 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 the owners.

Since M25, the target Reference Application codes involved in co-design have been identified as Alya, AVBP, CODA, 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).

2.1 Co-Design Management

The co-design task leader created a live spreadsheet to list all the cutting-edge platforms for co-design offered by the Core Partners, which is now maintained by Task 3.4's test lab for emerging technologies. The list of co-design platforms is augmented by the "EuroHPC JU Benchmark and Development Access Calls", which provides an access mechanism to even more cutting-edge platforms[2].

The leader of the Co-Design Working Group established and maintained an active collaboration with another CoE, namely CompBioMed [2], and attended their bi-monthly meetings when possible. Excellerat lead the Co-Design session at the First joint CoEs Technical Workshop,

[2] https://prace-ri.eu/hpc-access/eurohpc-access/eurohpc-ju-benchmark-and-development-access-calls/

Jan 2021, in collaboration with HiDALGO [3] and ChEESE [4]. The leader also represented EXCELLERAT at a round-table discussion of co-design amongst many CoEs, chaired by FocusCoE, and held within HiPEAC21.

Our living document, which started life as a Word document in BSCW, was moved to our internal wiki to avoid the debilitating "single-editor bottleneck", and has eventually been moved to the EXCELLERAT edocs repository, as the wiki was found to require overhauling to allow tables.

Finally, we had planned to organise a joint Birds of a feather (BoF) session with ETP4HPC [5] at ISC'21; however, this was postponed due to Covid-19. EXCELLERAT was invited to join the FocusCoE co-design workshop at ISC'22; however, EXCELLERAT participation depends on the organisers enabling virtual attendees.

The remainder of this Section on co-design 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 was created by Task 4.1, is maintained by Tasks 3.4 and is available in the edocs repository within the live spreadsheet. Regarding co-design, the hardware involved to-date are as follows, listed per HPC provider.

- TGCC
    - ARM Thunder X2
    - Fujitsu A64FX
    - AMD EPYC cluster at TGCC (Très Grand Centre de calcul du CEA).
- BSC
    - RISC
    - MN4 CTE-ARM
    - MN4 CTE-AMD
    - MN4 CTE-Power, Power9 + GPUs cluster
- CINECA
    - IBM Power 9 plus NVIDIA Volta GPUs, MARCONI100 (M100)
- DLR
    - AMR CPU and GPU clusters
- E4
    - ARM Marvell ThunderX2 and NVIDA Tesla GPUs, ARMIDA, from EPI
- EPCC
    - HPE Cray EX : ARCHER2
    - ARM Cavium ThunderX2 cluster from HPE
    - FPGA: Xilinx's Alveo U280 FPGA cluster
- HLRS
    - HPE Apollo 6500 Gen 10 Plus
    - HPE Apollo
    - NEC Vector machine: SX-Aurora
- CSC
    - LUMI
    - LUMI C

- IDRIS
    - HPE CPUs
    - HPE GPUs, JEANZAY
- FZJ
    - Juawei
        - Hauwei Cavium Thunder X2 from EPI
    - Jauwei
        - Hauwei Kunpeng 920 (ARM)

2.2 *Alya/BSC*

2.2.1 Cache-aware Sparsity Patterns for the Factorized Sparse Approximate Inverse Preconditioner in multi-node executions

Conjugate Gradient is one of the methods of choice for the iterative solution of symmetric and positive definite linear systems. Part of its effectiveness relies on finding a suitable preconditioner that accelerates its convergence. Factorized sparse approximate inverse (FSAI) preconditioners [6] [7] are a prominent option with the advantage that, relying on the sparse matrix-vector multiplication (SpMV) kernel for its application, they are easily parallelizable and portable to modern computational architectures.

An essential element of FSAI preconditioners is the definition of the sparsity pattern where the inverse is approximated. This definition is generally based on numerical criteria. In EXCELLERAT, BSC has introduced complementary architecture-aware criteria that also increase the computational efficiency of the preconditioner. In particular, we define cache-aware pattern extensions that do not generate additional cache misses when accessing the multiplying vector. In the third year of the project, we have particularly focused on applying this technique on distributed memory parallelisations, including the minimization of inter-node communications.

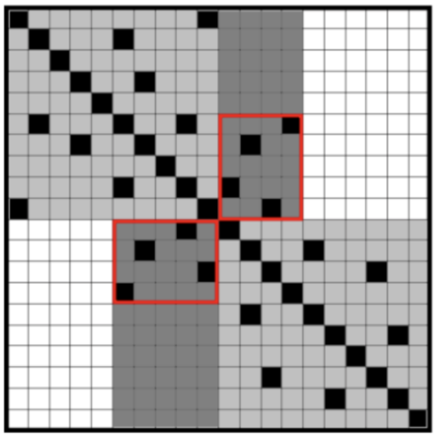


Figure 1: FSAIE-Comm. Graphical illustration of halo region where entries can also be added in a cache-friendly communication-aware extension. Black squares correspond to initial entries.

An illustration of this idea is shown in Figure 1, where new cache-friendly entries are added in the halo region that avoid incurring in additional communication requirements. The effectiveness of this methodology has been tested in a set of 39 matrices from the SuiteSparse Matrix Collection [8] [3]. We include experiments for symmetric positive definite (SPD) matrices with the number of non-zero entries ranging from 1M to 40M. Tests were performed on the CTE-ARM cluster composed of nodes with an ARM 48-core Fujitsu A64FX CPU. In this case, the CPUs have 256Bytes cache lines allowing large cache-aware extensions. On average, FSAIE-Comm allows to reduce iterations-to-convergence up to 31% and time-to-solution up to 26%.

Figure 2 displays the time-to-solution improvement for each matrix of the data set when using the best $Filter$ and when using a $Filter$ value of 0.05. Results show that all matrices of the data set take advantage of the proposed cache-friendly extensions, achieving in some cases a time-to-solutions decrease up to 60%.

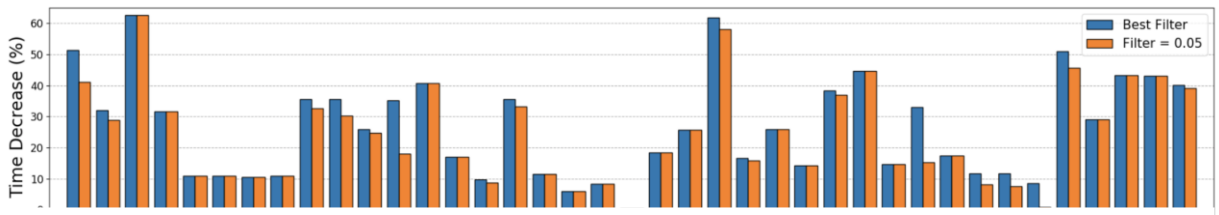


Figure 2: Time decrease of the FSAIE-Comm vs. FSAI for the best $Filter$ value (blue columns) and for the 0.05 $Filter$ value (orange columns) on the A64FX architecture.

2.2.2 Porting Alya to FPGA

We have begun the process of porting Alya to an FPGA cluster by first considering one of its modules, namely *nastin*. This process is presented in detail in D3.3 *Final Report on Exa-enabling enhancements and benchmarks* and summarised here. The module *nastin* is a key component of many Alya simulations, is used to model incompressible flow and can represent 64% of runtime for representative benchmarks.

Once we had obtained optimal single kernel performance, we explored additional benchmarks and multiple FPGA kernels. There are two versions running on the FPGA, one with the entirety of the matrix assembly code on the FPGA and the other with the partial kernel present placing the cartesian derivatives function on the CPU. The latter is helpful as whilst cartesian derivatives is responsible for less than 5% of the overhead of the kernel when profiling, it accounted for around 20% of the usage on the FPGA. Consequently, these two approaches limit us to either two or three matrix assembly engines (see D3.3 for details), for the entirety of the kernel or the partial kernel on the FPGA respectively.

For this work, we use a Xilinx Alveo U280 FPGA. For comparison, we run the code on a 24-core Xeon Platinum (Cascade Lake) CPU, and NVIDIA Tesla V100 GPU. On the CPU, the code has been parallelised via OpenMP (using GCC 8.3), while on the GPU it uses OpenACC (using NVIDIA compiler version 20.9). All reported numbers are averaged over three runs.

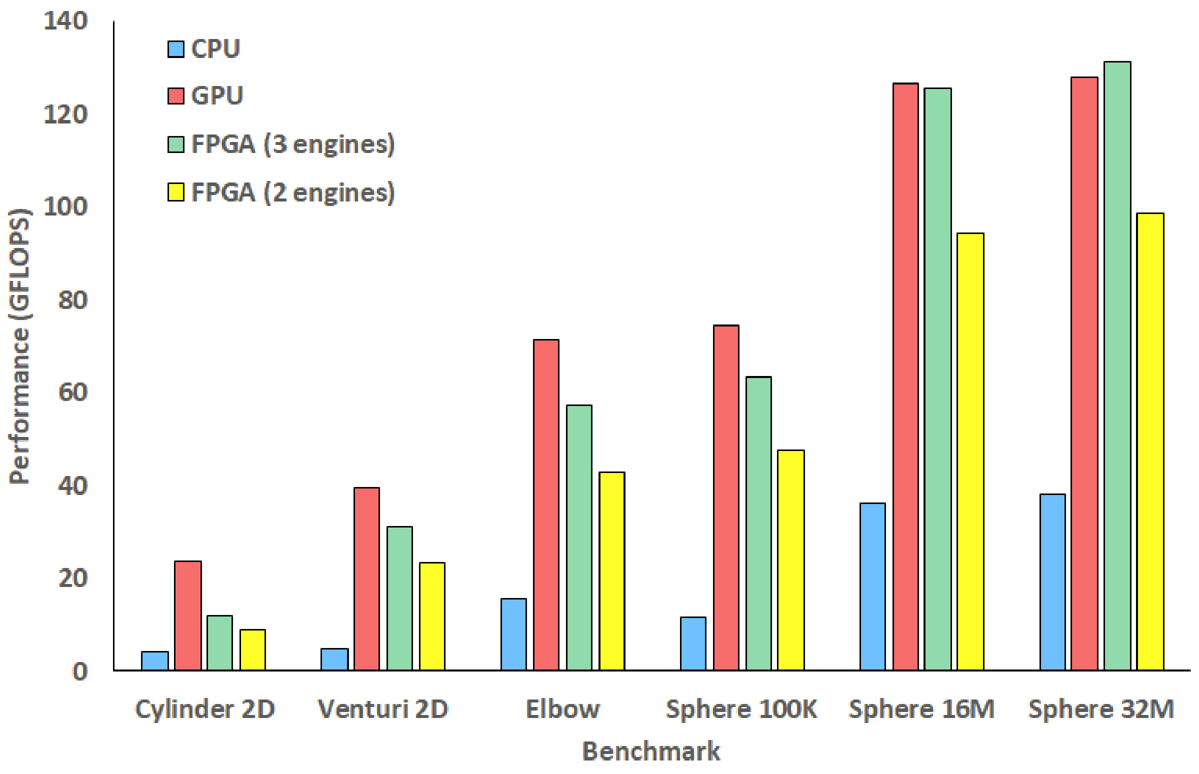


Figure 3: Performance comparison between CPU, GPU, three kernels on the FPGA (with cartesian derivatives calculation on CPU) and two kernels on the FPGA (all calculations on the FPGA). Higher is better (higher performance).

Figure 3 illustrates the overall performance for different benchmarks running on the 24-core CPU, V100 GPU and FPGA. It can be seen that the FPGA version of *nastin* is highly competitive against the GPU for larger problem sizes and consistently out-performs the CPU. The three engine FPGA approach is preferable to the two-engine approach due to the increased concurrency that it provides.

Figure 4 reports an average power draw of these different configurations. We only report numbers for the largest benchmarks as the others ran too quickly to obtain reliable power figures. The CPU draws the most power, while the GPU consumes slightly less but still more than 160 Watts for the largest benchmark. The FPGA power draw is significantly less than the other technologies, with three engines drawing 100 Watts for the largest benchmark and two engines drawing 70 Watts. The difference in FPGA power draw can mostly be ascribed to the CPU, which consumes more power for the three-engine approach as the cartesian derivative calculations are on the CPU. By contrast, even though the two-engine approach still processes input and accumulates the resulting data on the CPU, this results in significantly less workload than the threading floating point operations required by the three engine approach on the host.

Therefore, power consumption is much less.

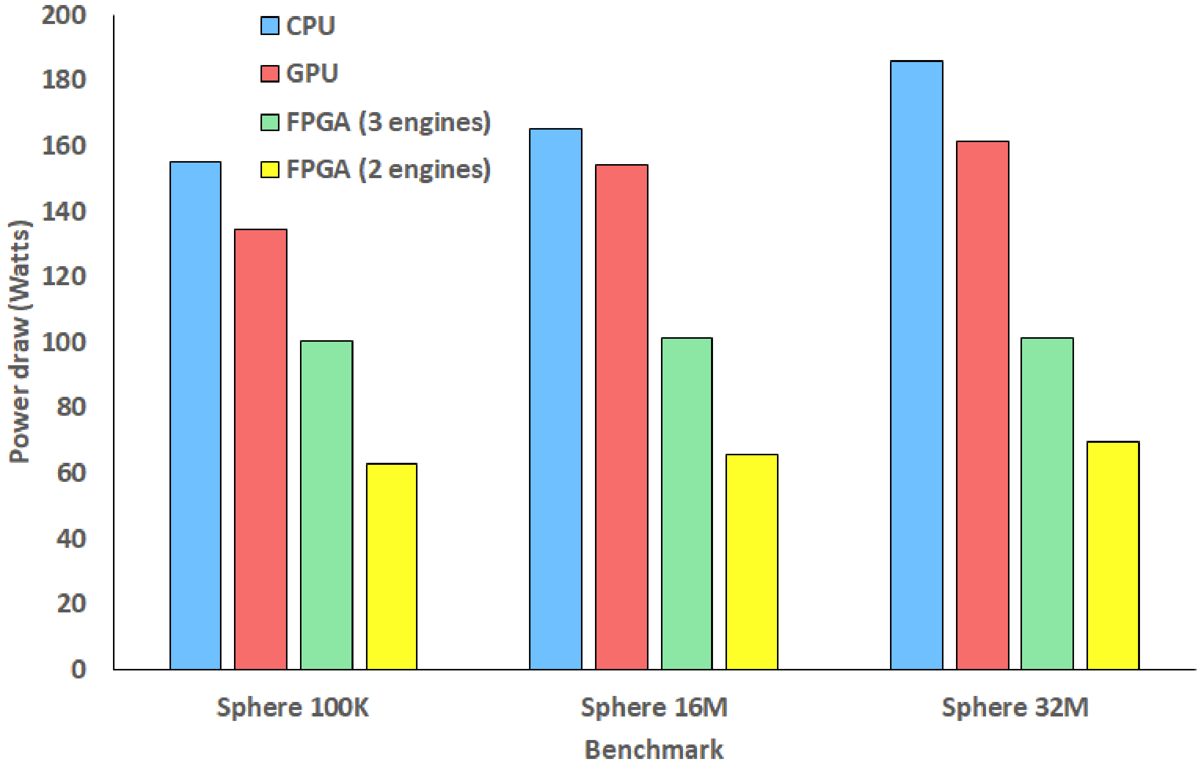
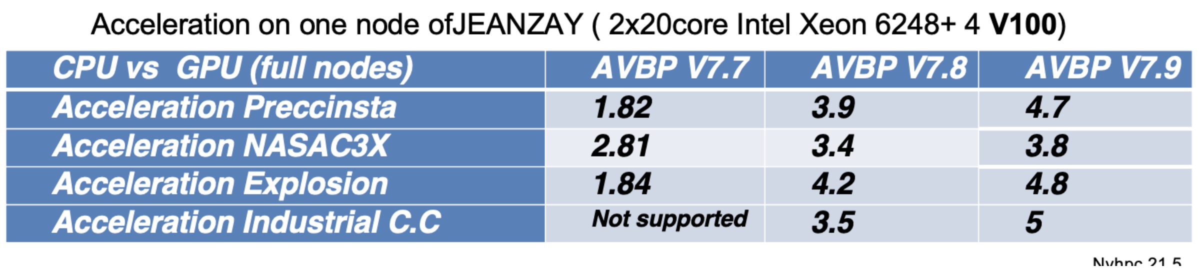


Figure 4: Power draw comparison between CPU, GPU, three kernels on the FPGA (with cartesian derivatives calculation on CPU) and two kernels on the FPGA (all calculations on the FPGA). Lower is better (less power draw).

2.3 AVBP/CERFACS

Co-design and performance optimisation have been focused on the whole AVBP code since D4.6. Two main axis of performance engineering have been studied: ARM-based architectures and GPU acceleration using NVIDIA hardware.

The first fully functional port of AVBP using OpenACC on NVIDIA hardware was released to the users in September 2020. Since then, focus has been maintained on performance optimisation and porting missing execution branches towards the completion of one of our use cases, namely the simulation of a full gas turbine combustion chamber on accelerated hardware.



Acceleration on one node ofJEANZAY (2x20core Intel Xeon 6248+ 4 **V100**)

CPU vs GPU (full nodes)	*AVBP V7.7*	*AVBP V7.8*	*AVBP V7.9*
Acceleration Preccinsta	*1.82*	*3.9*	*4.7*
Acceleration NASAC3X	*2.81*	*3.4*	*3.8*
Acceleration Explosion	*1.84*	*4.2*	*4.8*
Acceleration Industrial C.C	*Not supported*	*3.5*	*5*

Nvhpc 21.5

Table 1: Acceleration factor on full CPU versus 4 V100 GPUs.

Table 1 shows the acceleration factor achieved for each release since 7.7 until the release of 7.9 in Q4 2020. The industrial combustion chamber (C.C) corresponds to full gas turbine combustion chamber use case..

Since node level performance was excellent, we also ported and tested the GPU code on the JUWELS BOOSTER system at JSC equipped with A100 accelerators. Porting was automatic without any changes required to the code and yielded a 2-times-acceleration compared to V100 GPUs (See Figure 5).

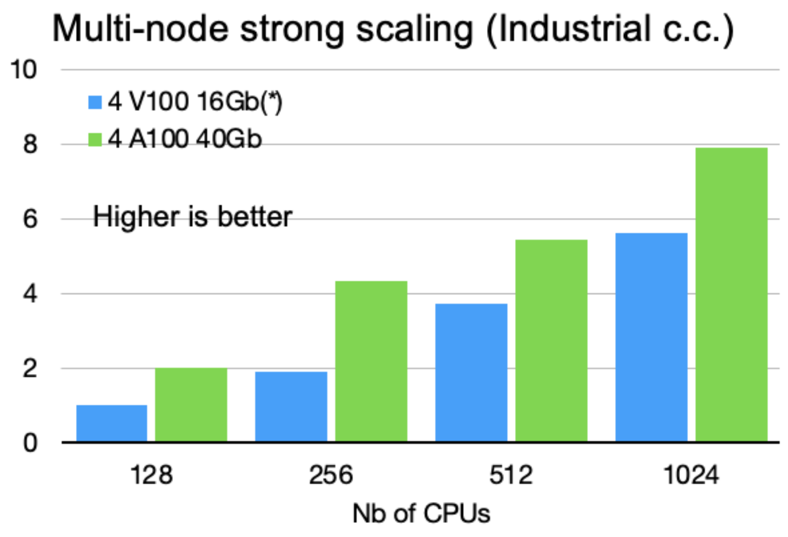


Figure 5: Strong scaling on V100 and A100 using the full gas turbine combustion chamber use case.

Future work will focus on adapting the existing accelerated workflows using OpenACC to a portable runtime such as OpenMP 5 to be compatible with AMD and Intel accelerators. These work has started using a co-design bilateral collaboration with Intel and Lenovo and will be continued in Excellerat 2 should the proposal be accepted.

In parallel to GPU, we focused on emerging ARM technology. The AVBP code was ported and benchmarked on the ARM Ampere and AWS Graviton 2 architectures in collaboration with ARM Ltd and AWS. (See Figure 6).

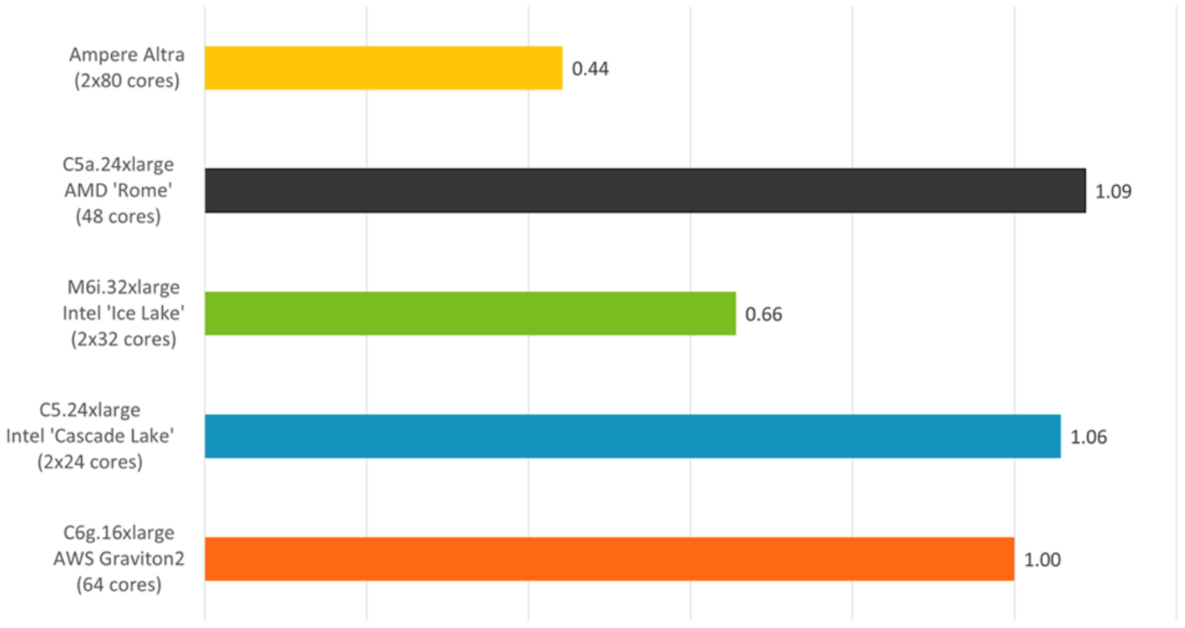


Figure 6: Normalized timing (graviton 2 performance as reference) on the full gas turbine combustion chamber use case using a static mesh and a single full node for each architecture.

These results confirm the view that ARM-based architectures are almost on par with traditional Intel and AMD chips.

2.4 CODA/DLR

To test CODA's computational kernel on novel architectures and accelerators, a proxy has been created, thereby removing the need to adapt the entire workflow of CODA. This proxy 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 first focus during the reporting period (since D4.6) was on the performance evaluation and optimization on GPUs. After the numerical stability and correctness was verified, the initial performance was evaluated on up to 16 NVIDIA V100 GPUs and a detailed performance analysis was carried out to identify potential performance bottlenecks. After that, Spliss has been optimized and re-analyzed multiple times in a co-design cooperation with NVIDIA. Optimizations include among others the inclusion of CUDA-aware MPI and GPUDirect, the improvement of host-to-device copies and the inclusion of CUDA multi-process service (MPS) to allow more flexibility in the allocation of CPUs and GPUs.

Finally, Spliss on GPUs was evaluated again. The current version achieves a speedup of 4.2 to 4.4 in a node-wise comparison of two Intel Xeon 6230 (40 cores) vs. 4 NVIDIA V100 GPUs, which is a speedup of 2.4 to the initial version. In addition, Spliss was evaluated on the JUWELS BOOSTER System at JSC and achieved a very good parallel efficiency of 63% on 128 NVIDIA A100 GPUs.

AMD cluster

The second focus during the reporting period was on the scalability evaluation and optimization on DLR's main HPC system CARA, which is based on the AMD Naples architecture. The used test case solves the Reynolds-averaged Navier-Stokes equations (RANS) with a Spalart-Allmaras turbulence model in its negative form (SA-neg). The test case runs on an unstructured mesh from the NASA Common Research Model (CRM) with about 5 million points and 10 million volume elements. The mesh is rather small, which has been chosen for a strong scalability analysis of Spliss (fixed problem size). Production meshes would be at least 20 times larger and accordingly achieve comparable efficiency on much higher scales.

During the reporting period, the scalability of Spliss has been further improved: For the rather small mesh employed, Spliss achieves about 71% parallel efficiency on the largest available partition on CARA with 512 nodes and 32,768 cores. Before EXCELLERAT, Spliss achieved 60% parallel efficiency on CARA.

2.5 Nekbone (Nek5000)/CINECA,KTH

The Reference Application Nek5000 has a mini-app entitled Nekbone [9] which is used for co-design.

2.5.1 HPC architectures at CINECA and E4

Two different test campaigns have been conducted on the Nekbone code. The first campaign involves a comparison between different machines and architectures without accelerators. The tested machines were Galileo (Intel Broadwell 36 cores), Marconi A3 (Intel Skylake 48 cores), Marconi 100 (M100) on Power9 CPUs (IBM Power 9), and ARMIDA (E4 ARM 64).

The results obtained with Marconi A3 and Galileo are very similar, and hence only results from Marconi A3 will be shown. On ARMIDA, different compilers (ARM and GCC) have been tested and no significant differences have been found.

The second test campaign involves only M100 with accelerators (NVIDIA V100 GPUs), where weak and strong scalability tests have been performed with different strategies for the code porting.

Nekbone is the proxy distilled from Nek5000 to the aim of reproducing the behaviour of its principal computational kernel. The target of this code is to test the performance without the overhead of the whole software. Nekbone is written mainly in fortran77 and is a spectral (high order) fluid-dynamic solver for the uncompressible Navier-Stokes equations.

The parameters involved in this test campaign are nx=10 and 16, where nx is the polynomial degree, and nelem=1024 and 3584 where nelem is the number of elements assigned to each GPU.

The first campaign compared the performance with the single node results in [10], performed on Kebnekaise,, a machine at HPC2N in Sweden with a 28-core Skylake CPUs. In [10], one conclusion is a recommendation to use 16 to 128 elements per core in order to achieve optimal performances. Note, 9th order polynomials (nx=10) have been used.

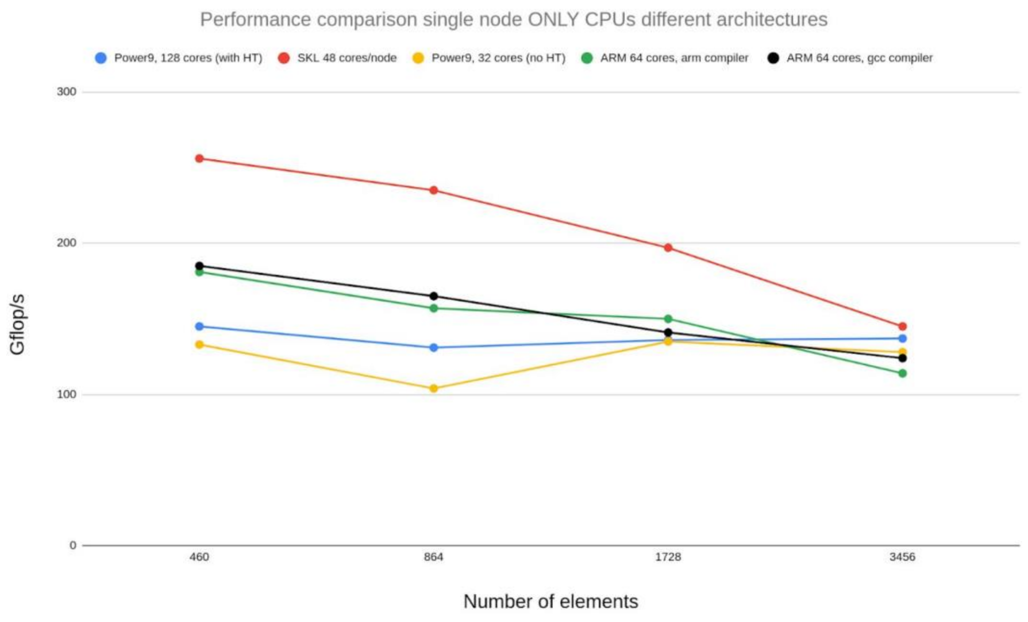


Figure 7: Single node results obtained during CINECA test campaign.

Figure 7 shows the results obtained by CINECA, which show higher Gflops than on Kebnekaise, labelled as "SKL 48 cores/node", as reported in [10]. Moreover, Marconi A3 (Power9, SKL 48 cores) is the best performing machine for any number of elements.

The first campaign continued with the weak scaling tests using CPUs, performed by keeping constant the number of elements per core (128 per node). Results are displayed in Figure 8.

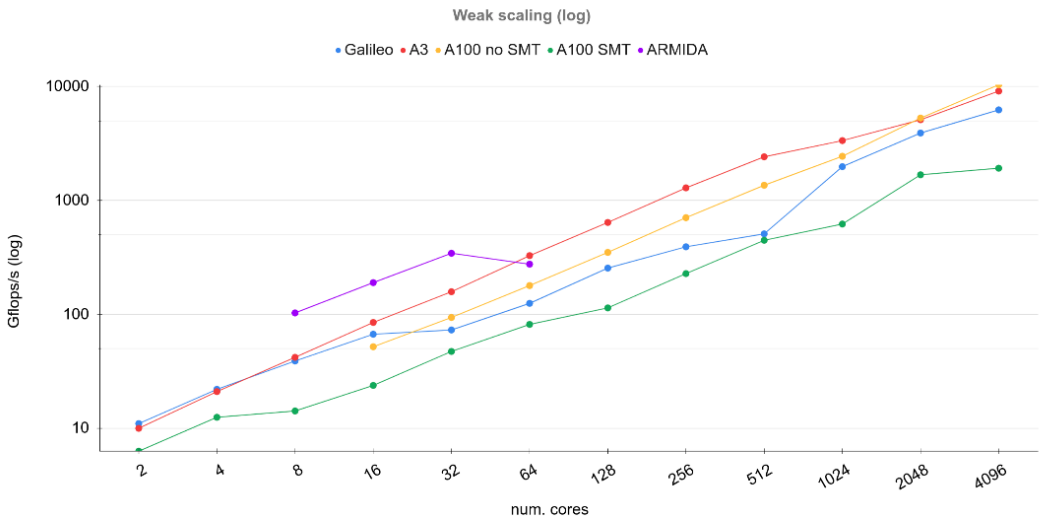


Figure 8: CPU-only multi-node weak scaling.

As it can be seen from Figure 8, the best performances have been achieved with M100 with SMT disabled. The results here are similar to [10].

The second campaign with weak scaling tests on a single GPU node, performed on M100 using 3 different GPU porting strategies: OpenACC, OpenACC + CUDA Fortran (only for PGI compiler), and OpenACC + CUDA C. All these tests have been performed on a single node, using 1 GPU accelerator per MPI task, with 1 and 4 tasks. A "*better performance for higher programming effort trend*" is clearly visible in **Fehler! Verweisquelle konnte nicht gefunden werden.** and Figure 10: Best performances have been obtained with OpenACC + CUDA C.

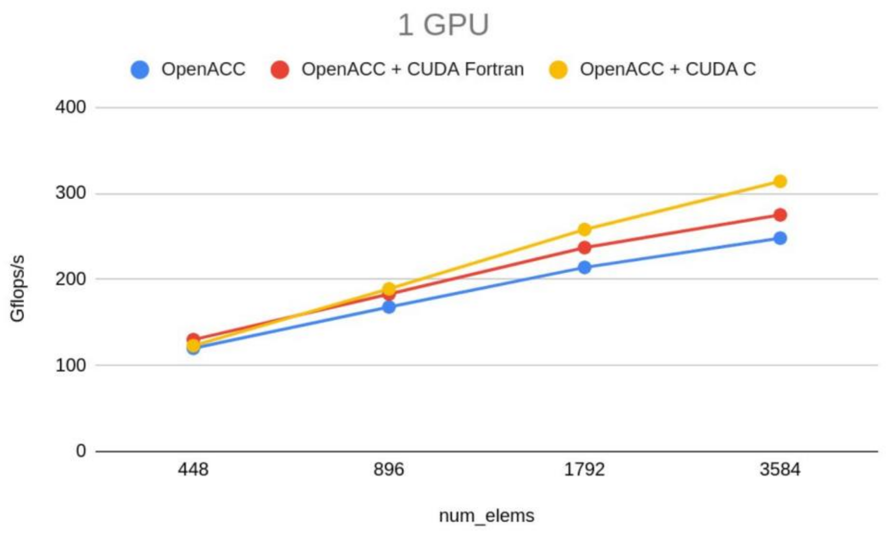


Figure 9: Weak scaling single-node with 1 GPU.

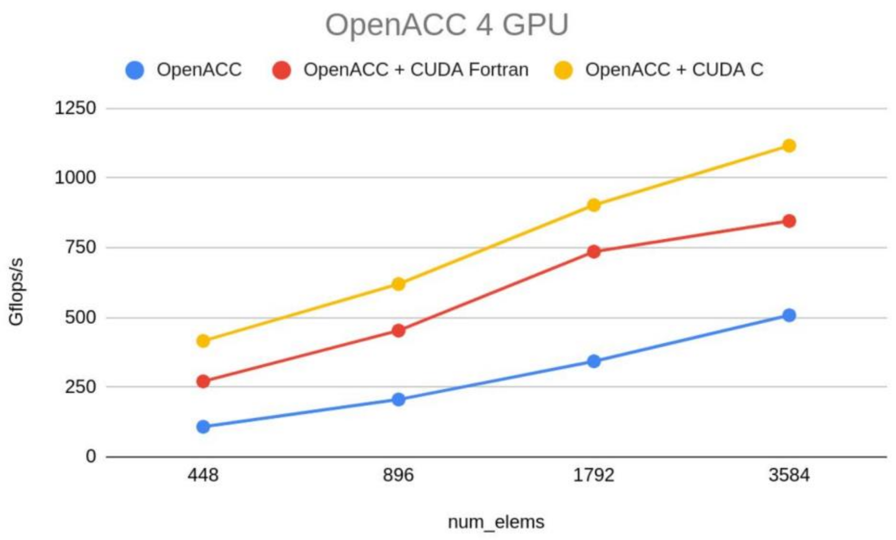


Figure 10: Weak scaling single-node with 4 GPUs.

The most critical part of the porting is the mapping between MPI processes and the GPUs, which has been enforced directly in the code to achieve results similar to [10]. A non-optimal mapping can reduce performance by 2 orders of magnitude. The performances obtained are slightly lower than [10], but can be improved via tuning simulation parameters.

On multi-node tests: with nx=16 and nelem=1024, where it was seen that "OpenACC" and "OpenACC + CUDA Fortran" both scaled very well, with similar performance; however, nx>10 is usually not employed in real-world cases. Indeed, "OpenACC + CUDA C" does not run for nx>10. Tests with nx=10 have been performed (Figure 11); here, performance is lower than with nx=16. Moreover, "OpenACC" shows lower scalability above 256 GPUs.

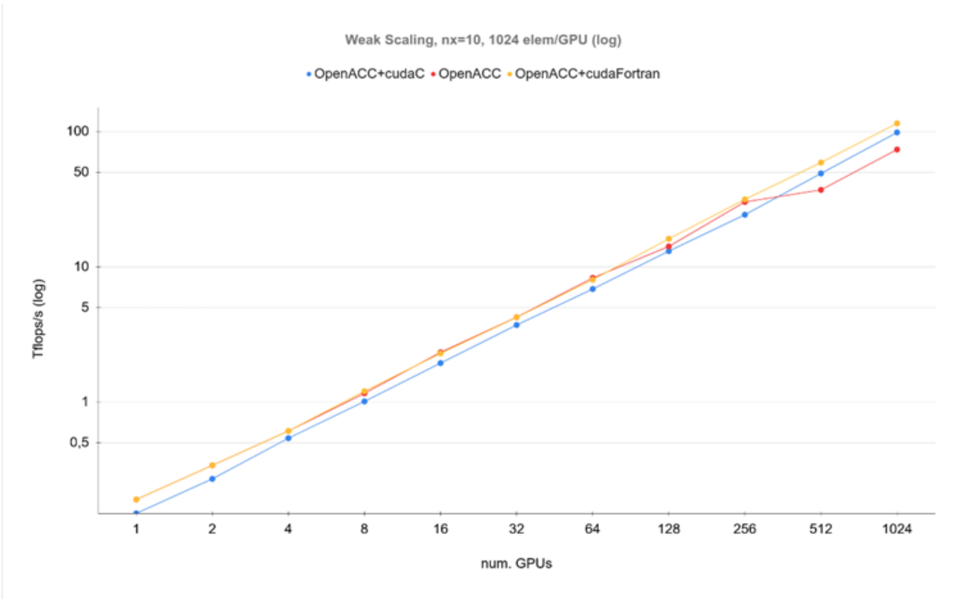


Figure 11: Weak scaling multi-node (polynomial degree 9, elements per GPU 1024).

To improve the overall performance, computational intensity was increased by increasing nelem to 3584 (See Figure 12). It was noted that nx=10 with nelm=3584 gave almost the same performance as nx=16 with nelem=1024. The pure OpenACC version was found to be unstable for nelem=3584, and hence the results have been omitted in Figure 12.

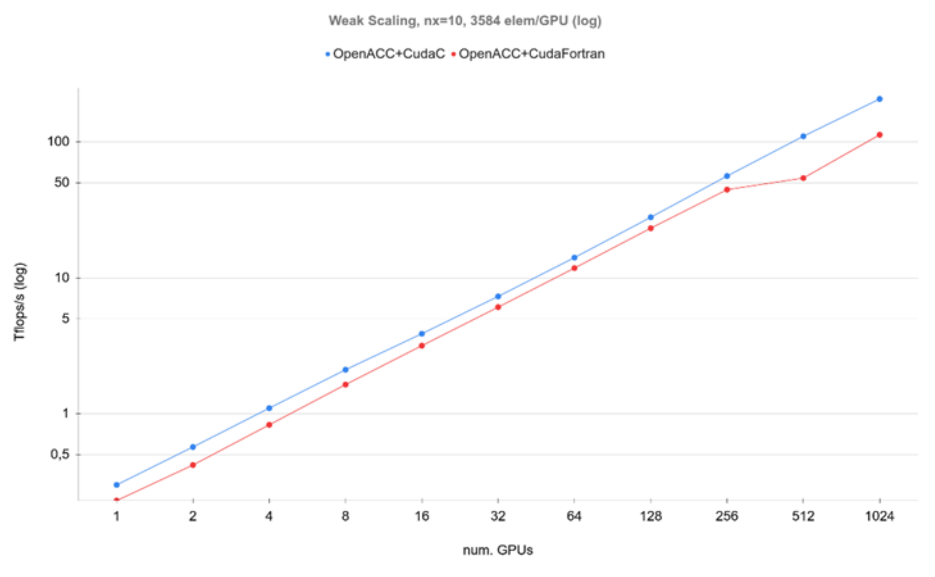


Figure 12: Weak scaling multi-node (polynomial degree 9, elements per GPU 3584).

Figure 13 shows results for "OpenACC" with nx=16 and nelem=1024 and "OpenACC + CUDA C" with nx=10 and nelem=2584. The entirety of the M100 machine was employed. The simulation reached a peak of about 800 Teraflops on 3792 GPUs. Results from "OpenACC + Fortran C" were found to be unstable and are not shown. The two approaches have similar performance, and both start deviating from linear behaviour at above 1024 GPUs.

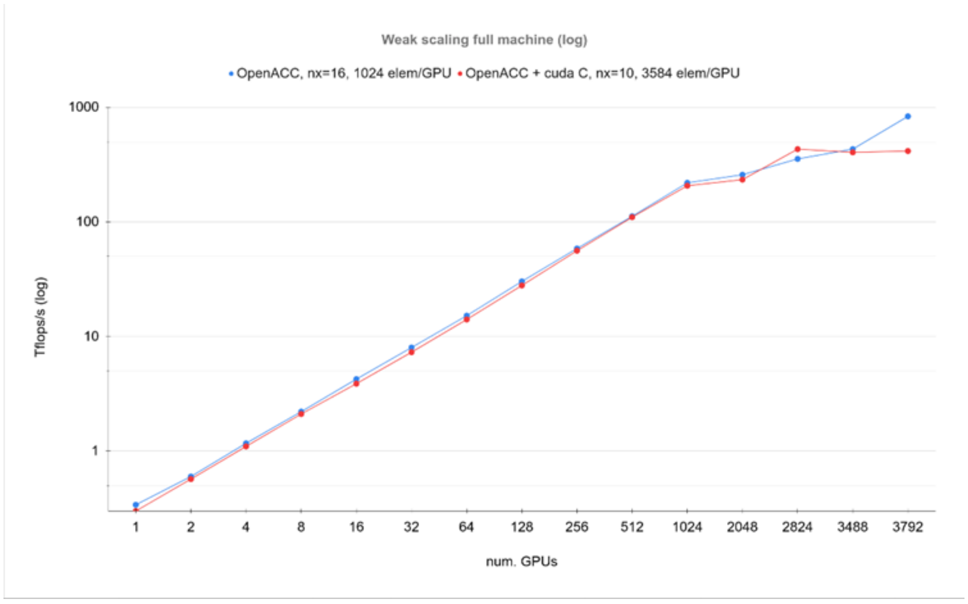


Figure 13: Weak scaling M100 full machine.

Finally, for strong scaling tests, the GPU count tested runs from 32 up to 1024. The lower bound is the lowest possible for the target simulation, while the upper bound is employed, as all codes became unstable when employing higher GPU counts.. As it can be seen from Figure 14, the three approaches are similar, and the scalability is approximately linear.

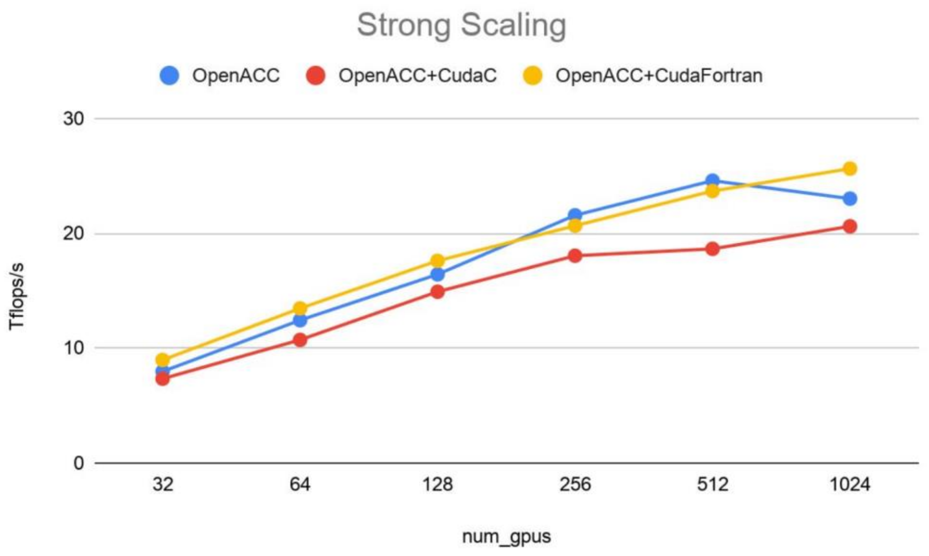


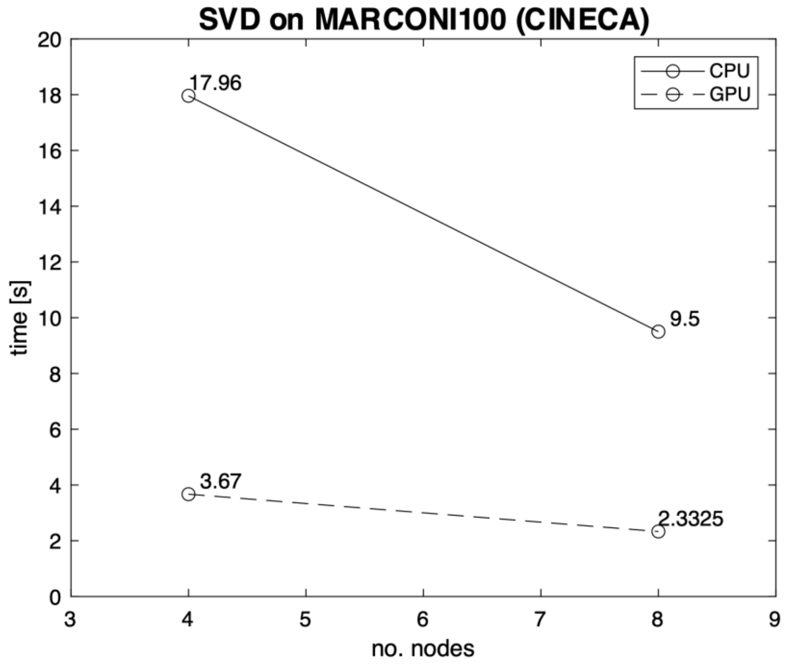
Figure 14: Strong scaling multi-GPUs.

2.6 ScaLAPACK/RWTH

RWTH developed a user-friendly modal decomposition toolkit to efficiently perform dynamic mode decomposition (DMD) and proper orthogonal decomposition (POD) on pre-Exascale HPC systems in EXCELLERAT.

The computationally most expensive part of the modal decomposition algorithms is an economy-sized singular value decomposition (SVD) of the snapshot matrix. The snapshot matrix is a tall (“skinny”) matrix containing the time-resolved flow field data at each mesh point of the flow field with a size of up to O(10^9) times O(10^3) entries for large scale investigations. To guarantee an efficient postprocessing workflow using the modal decomposition toolkit, minimising the computational time required for the SVD is essential. To this end, different SVD algorithms have been implemented and tested as described in Section 4. The implementation of the various SVD algorithms is based on the LAPACK [11] and ScaLAPACK [12] libraries. In order to test the performance on different hardware systems, a proxy was developed, which encapsulates the SVD from the rest of the modal decomposition algorithm,. The proxy was ported to the IBM-based M100 cluster at CINECA. To guarantee the best performance, the original LAPACK and ScaLAPACK implementations were replaced by the IBM-specific Engineering and Scientific Subroutine Library (ESSL). To further reduce the computational time, the proxy was ported to GPUs using the MAGMA Linear Algebra Library. In more detail, the core part of the SVD algorithm, which uses an eigenvalue decomposition of the cross-product matrix (see Section 4), was shifted to the GPUs leading to a fourfold increase in speed as illustrated in Figure 15.

The investigations revealed that a strong limiting factor for the SVD on GPUs is the restriction of the available memory. This restriction can possibly be reduced by a currently developed library called SLATE [13] which will be able to run on distributed-memory environments extending the ScaLAPACK library to GPUs.



M100:

2 x IBM POWER9 AC922 per node

- 16 cores @ 3.1 GHz per CPU
- 256 GB RAM per node

4 x NVIDIA Volta V100 GPUs per node (16 GB)

Figure 15: Execution time used by the SVD via eigenvalue decomposition of the cross-product matrix on CPUs and GPUs

2.7 Summary of Co-Design Activity

The WP-transversal Co-Design Working Group and the Co-Design Task (Task 4.1) have contributed to EXCELLERAT's Advanced Services and have produced the Consulting Service: Co-Design Service for Engineering Applications. On the software co-design side, based on the Reference Applications and other relevant applications and libraries of the engineering workflow, the operations, kernels and algorithmic features have been characterised. These are common and widely used in engineering applications demanding large amounts of computational time. On the hardware side, we are cooperating 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 have maintained our agreed methodology, as stated in the Introduction to this Section, wherein the client's application (or proxy) is 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. As described in this Section five applications have been exploiting our Co-Design Service, two of these having a proxy versions (one created since M24). As described above, four of the applications have now been ported to a number of novel machines, including CPU clusters, GPU clusters, and an 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, namely Alya and Nekbone, by CoE members at other core partner sites. For the latter case, the required code adaptations were fed back to the code owners.

Finally, we are proud to continue to provide consulting for software and systems co-design. This service offering has been improved and is currently available within the Co-Design Engineering Software- and System-Design service (see D4.2 and D1.6, *Reports on the Service Portfolio* for a more detailed description).

3 Task 4.2: Visualization

In order to focus on I/O on a manageable level and to supervise simulations during runtime, the main goal of this task has been set as the enablement of in-situ post-processing capabilities in selected applications.

One tool chosen for this purpose is Vistle [14] for which two in-situ interfaces have been developed. The first one is the LibSim interface used by VisIt [15] as outlined in D4.2. The second interface is obtained via the SENSEI [16] framework. In D4.2 we described an approach made using Catalyst [17], 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 and Nek5000 was instrumented with SENSEI.

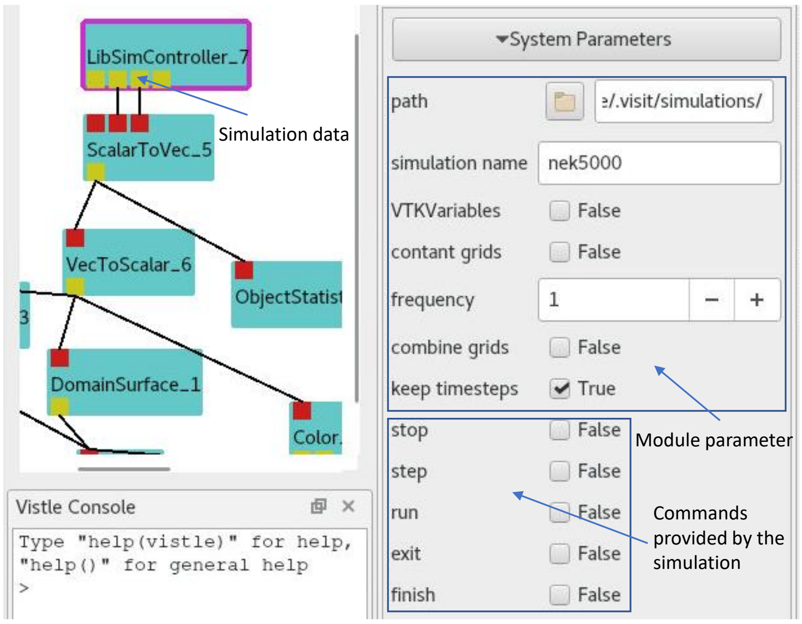
While Vistle is focused on interactive visualization in 3D virtual environments, the other post-processing tool developed in this project is a High-Performance Computing analysis tool (PAAKAT). This tool allows a straightforward real-time handling of data arisen from simulations with pre-configured algorithms based on the Visualization Toolkit (VTK) [18].

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 data allocated outside of Vistle. 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 16. 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 at simulation run-time 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.



```
Type "help(vistle)" for help,
"help()" for general help
>
```

path	ə/.visit/simulations/
simulation name	nek5000
VTKVariables	False
contant grids	False
frequency	1
combine grids	False
keep timesteps	True
stop	False
step	False
run	False
exit	False
finish	False

Figure 16: Connected LibSimController-module in Vistle.

3.1.1 LibSim interface for Vistle

The LibSim interface consists 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 socket 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. At the start of a simulation, used back-ends can be configured 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).

3.1.3 In situ Visualization with AMR version of Nek5000

Due to the developments on Nek5000 described in D2.4 *Final Report on Reference Applications Outcomes*, the existing LibSim interface is no longer compatible with the up-to-date version of Nek5000. For example, LibSim does not support adaptively refined grids. Therefore, VTK-based interfaces have been implemented for Nek5000 to allow in-situ analysis with the developed Nek5000 version. A Catalyst interface was developed to generate pre-configured images. For interactive use with Vistle, the SENSEI interface was also integrated.

First, to utilize SENSEI, the build system for Nek5000 had to be changed. Nek5000 ships with its own make file generator scripts that manage the compilation and linking. Since the Nek5000 repository usually self-contains all Nek5000 dependencies, this works fine. Linking to SENSEI and its dependencies, which themselves depend on the activated analysis backends, is way more complex. Therefore, a CMake-based build system generator was introduced to help locating the additional dependencies.

Once linked to SENSEI, the Nek5000 implementation of the SENSEI data adaptor updates pointers to the data fields of the Nek5000 simulation once per cycle. Then, control is passed to the analysis adapters. Only if they request data, data conversion to VTK is performed.

For backends such as Vistle, there seems to be no way to get information from a simulation through SENSEI, once a mesh has been refined (the transfer can only be optimized for completely static meshes). While a cached mesh is reused inside of Nek5000 if the mesh did not change, the backends still have to treat every received mesh as if it were new. This causes some extra overhead for Vistle, since the meshes have to be transformed and copied to Vistle's data representation.

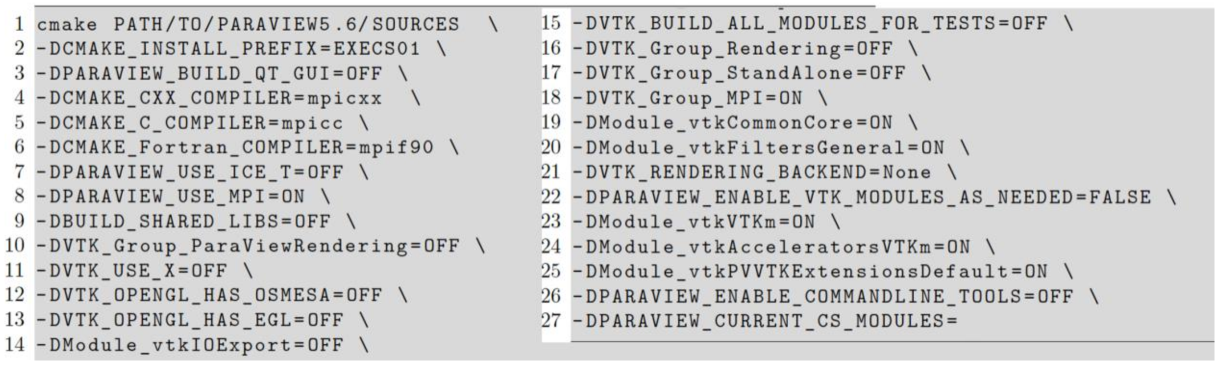
First runs of a pre-production version of the Nek5000 rotor case (C1U3) (described in D2.2 *Report on Reference Applications Outcomes*) with 346,336 total quadrants have been executed on the HPE Apollo (Hawk) at HLRS, using 8 nodes and 256 MPI processes. A cutting surface as well as in iso surface of the lamda2 domain have been generated in-situ with Vistle and streamed to a visualization cluster at HLRS to render.

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 in 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 [19] source code (version 5.6). Modifications performed in ParaView encourage the use of the C++ VTK API. As a consequence of these modifications, filters must be implemented by using C++ instead of the Python [20] scripts created by ParaView.

Using the whole ParaView framework, compilation times became a real issue. Tests on four nodes of MareNostrum 3 at BSC [21] with a total of 64 cores resulted in a compilation time of about 1,5 hours. By obviating rendering and therefore avoiding the need of big third-party libraries like Python and/or OpenGL [22], the compilation time was reduced to only nine minutes.

A list of the compiler options used to achieve this result are shown in Figure 17. These options have been successfully used in two supercomputers, Beskow at KTH and Nord III at BSC. As future work, more computer systems and compilers should 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 17: Compiler options for modified ParaView 5.6.

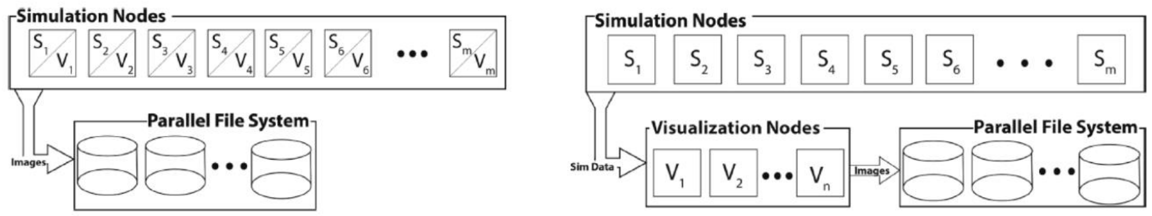
3.2.1 MPMD-VTK example

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 turbulence 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 4.3.

The problem can be solved in three main steps. First, interpolation is performed, then the interpolated data is extracted from ParaView, and thereafter used 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 studied in Task 4.2 (see Figure 18). In the problem that is being tested here, the usage of an 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 any 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 18: Parallel programming models. (a) Single-Program, Multiple-Data (SPMD) or in-line visualization. (b) Multiple-Program, Multiple-Data (MPMD) or in-transit visualization [23].

As mentioned above, another interesting point is related to the load balance. Figure 19 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 while an SPMD execution model is used.

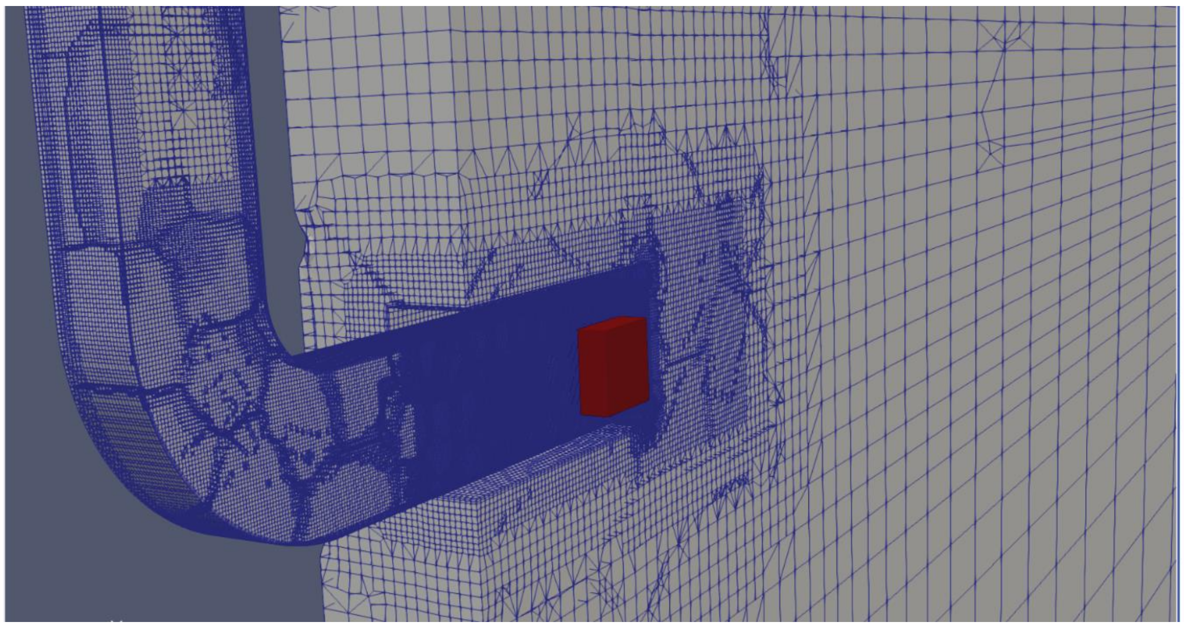


Figure 19: Source mesh (blue and grey) of 4.67e6 cells and volume of 4.43x4.09x0.22 cm^3. Destination mesh (red) of 1.25e5 cells and volume of 0.05x0.05x0.05 cm^3.

In order to overcome this drawback, an example using the MPMD model has been prepared (see Figure 20). 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.

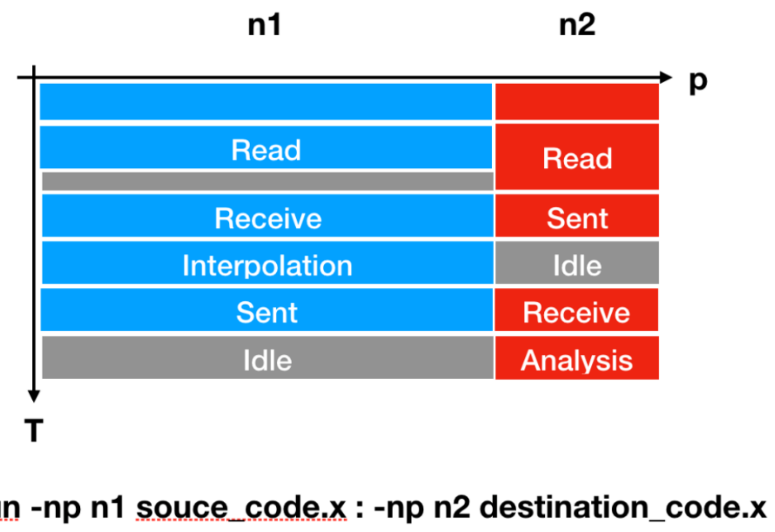


Figure 20: MPMD execution model of source code and destination code through n1 + n2 processors p.

3.3 Summary of Visualization Activities

The main focus of Task 4.2 has been to develop post-processing tools and workflows for exascale engineering simulations. Next to consulting in visualization topics and training offerings (see D5.7 *Final Services, Training and Portal Report*), the main effort of the visualization task is the development of scalable in-situ post-processing tools. Among the main challenges, we can mention 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.

The core-code Nek5000 has been successfully run in-situ with Vistle by using its existing LibSim interface or the implemented SENSEI interface with support for AMR. Due to the described issues with the production case, no measurements of the overhead could 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 developed a software framework for the purpose of in-situ analysis of CFD (computational fluid dynamics) data. Some key requirements have been identified which have been 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 21. 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 [24]. In order to take advantage of existing libraries, this was also a fundamental design strategy for our software toolbox. The core API follows the definition by scikit-learn [25] and thus enhances a simple integration of our code with external methods into a single processing pipeline.

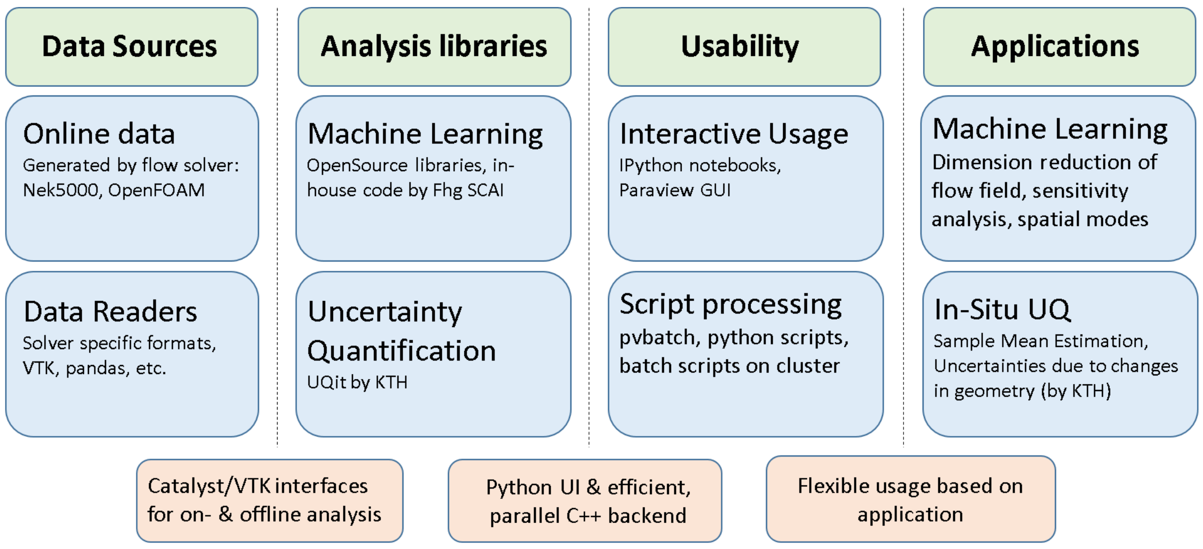

Data Sources
Analysis libraries
Usability
Applications
Online data
Generated by flow solver: Nek5000, OpenFOAM
Machine Learning
OpenSource libraries, in-house code by Fhg SCAI
Interactive Usage
IPython notebooks, Paraview GUI
Machine Learning
Dimension reduction of flow field, sensitivity analysis, spatial modes
Data Readers
Solver specific formats, VTK, pandas, etc.
Uncertainty Quantification
UQit by KTH
Script processing
pvbatch, python scripts, batch scripts on cluster
In-Situ UQ
Sample Mean Estimation, Uncertainties due to changes in geometry (by KTH)
Catalyst/VTK interfaces for on- & offline analysis
Python UI & efficient, parallel C++ backend
Flexible usage based on application

Figure 21: Software architecture for in-situ data analysis [26].

Besides flexibility, efficient implementations of the algorithms are a major requirement in the context of large-scale data analytics. Therefore, we kept the Python layer as thin as possible and built 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 [27]. 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 [28] for selected algorithms.

For in-situ data analysis, some of the methods have been wrapped as VTK [18] filters. In this way, they can be executed online in a Catalyst [17] pipeline or offline as ParaView [19] plugins. The latter also enables the import into a Python environment that provides interactive access to all VTK functionalities and integration with other analytics libraries in a client/server setup.

4.1.2 Applications

In a collaboration between KTH and Fraunhofer, in-situ algorithms for the estimation of the uncertainty due to finite time sampling have been implemented. This includes also a novel approach based on the modelling of the autocorrelation function during runtime. The approach has been applied to the entire three-dimensional velocity field of the flow around an airfoil, of which a slice is shown in Figure 22. A strong reduction of data I/O could be achieved while maintaining a similar accuracy compared to state-of-the art offline approaches based on autoregressive models. 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 a predefined threshold has been reached.

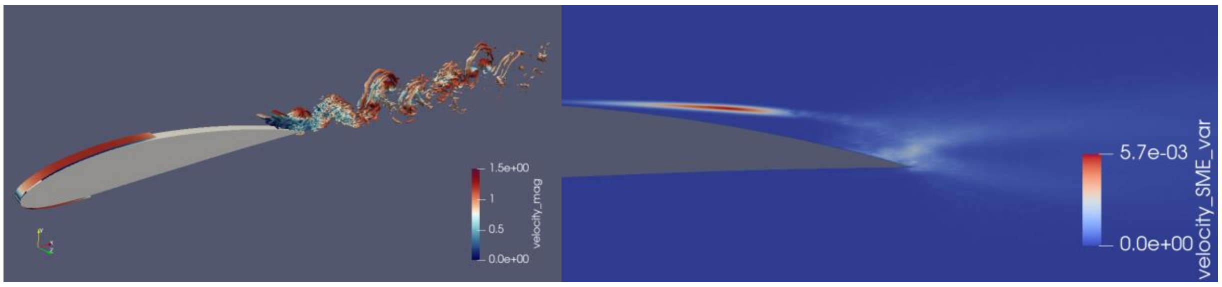


Figure 22: Flow around the NACA4412 computed with NEK5000; a) Iso surfaces of the Q-criterion of the instantaneous flow field b) Sample Mean Estimation (SME) at the midplane.

In addition, performance and scaling tests of the entire in-situ workflow have been performed. The strong scaling results are shown in Figure 23. For a low number of MPI ranks and therefore large local matrices, we observe only a small increase in the overall computing time of less than 5% compared to the overall simulation. This demonstrates the efficiency of our updating algorithm and framework in general, and its suitability as part of a practical simulation workflow. If the local mesh sizes get too small, a reduced performance and high memory usage can be observed due to the VTK and Python overhead in our prototyping framework. In this case, an implementation of the algorithm in pure C or Fortran could promise a further speedup. Considering the main focus of the framework being to support the flexible implementation and evaluation of new algorithms in Python, the performance results are more than satisfying.

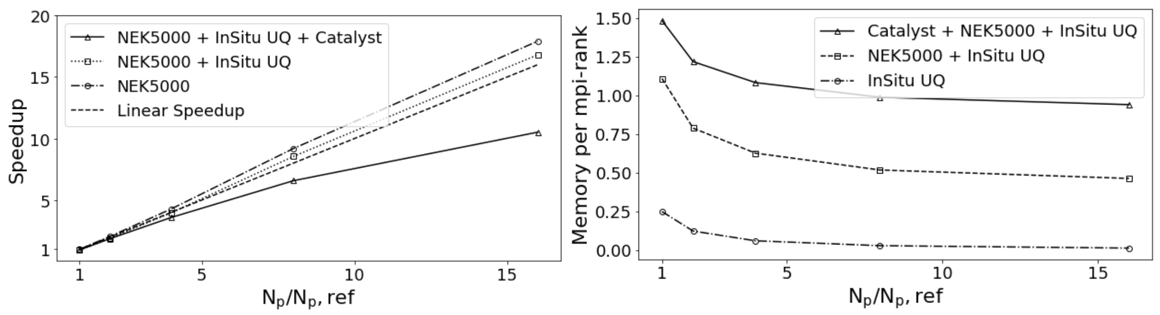


Figure 23: Strong scaling behaviour of the a) computing time and b) memory consumption for different numbers of cores.

In a second example, we computed the Principal Component Analysis (PCA) of the velocity field 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 this approach lies in earlier availability of the results. This can be exploited to generate a low-dimensional representation of the flow field during run time with the goal of monitoring the progress of the solution. In contrast to pure scalar quantities, such as local flow variables or numerical residuals, a more global view on the data is possible that can also capture new phenomena in a flexible way. By comparing the current simulation to previous runs, we are able to detect outliers that show unphysical or unwanted behaviour, such as unfulfilled constraints or errors in the numerical setup. Furthermore, a Machine Learning based clustering of the simulations can help to identify similar simulation results as an input for further analyses, such as sensitivity analysis or optimization. That way, we can save computing time by avoiding unnecessary runs as well as increase the accuracy of results by accounting for strongly non-linear effects related to a change in the input parameters. Results of the simulation monitoring and clustering will be presented at the ECCOMAS 2022 conference including a full paper submission to the conference proceedings [29].

4.2 Uncertainty Quantification

The development of UQit, a Python package for uncertainty quantification (UQ) in CFD has reached a good level. The first version of this open-source package has been recently released along with a journal paper published at JOSS (Journal of Open-Source Software) [26]. UQit is designed to be non-intrusively linked to any CFD solver, conditioned on having the appropriate interface. From the programming point of view, the design is flexible so that UQit can be efficiently extended in future. An extensive documentation of UQit [3]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, and a clear plan for continuous development of UQit is in place.

The 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

[3] https://kth-nek5000.github.io/UQit/

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. Samplingfor 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 assess UQ-based metrics for accuracy, sensitivity and robustness in CFD are described in our recent publication [30]. A novel aspect of the framework is to provide the techniques required for combining uncertainties from variation of inputs and parameters with observational uncertainties, see Figure 24. This is, in fact, consistent with the characteristics of high-fidelity turbulent flow simulations. The developed framework is used to evaluate performance of two popular open-source CFD software (Nek5000 and OpenFOAM) for simulation of canonical wall turbulence. In particular, impact of numerical parameters which include grid resolution in wall-parallel directions (for both solvers) as well as high-pass filtering parameters (for Nek5000) are thoroughly investigated in [31].

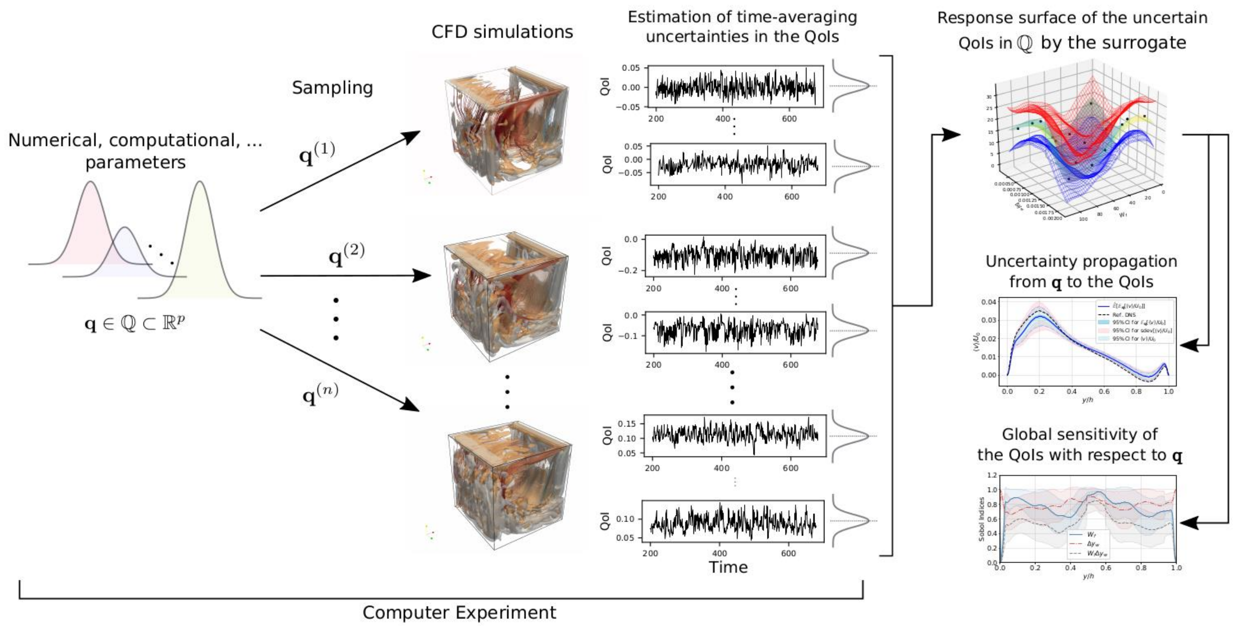

CFD simulations
Estimation of time-averaging uncertainties in the QoIs
Response surface of the uncertain QoIs in $\mathbb{Q}$ by the surrogate
Sampling
Numerical, computational, ... parameters
$\mathbf{q}^{(1)}$
$\mathbf{q}^{(2)}$
$\mathbf{q}^{(n)}$
$\mathbf{q} \in \mathbb{Q} \subset \mathbb{R}^p$
QoI
Time
Uncertainty propagation from $\mathbf{q}$ to the QoIs
Global sensitivity of the QoIs with respect to $\mathbf{q}$
Sobol indices
y/h
Computer Experiment

Figure 24: The workflow of the UQ analyses taken from [29]: A set of Category-I factors q ∈ Q with known distributions is given. Associated with each of the n samples taken for q, a CFD simulation is performed where the resulting QoIs (responses) of each simulation can be uncertain due to Category-II parameters. A practical example of the latter can be the uncertainty in the QoIs due to finite time averaging.

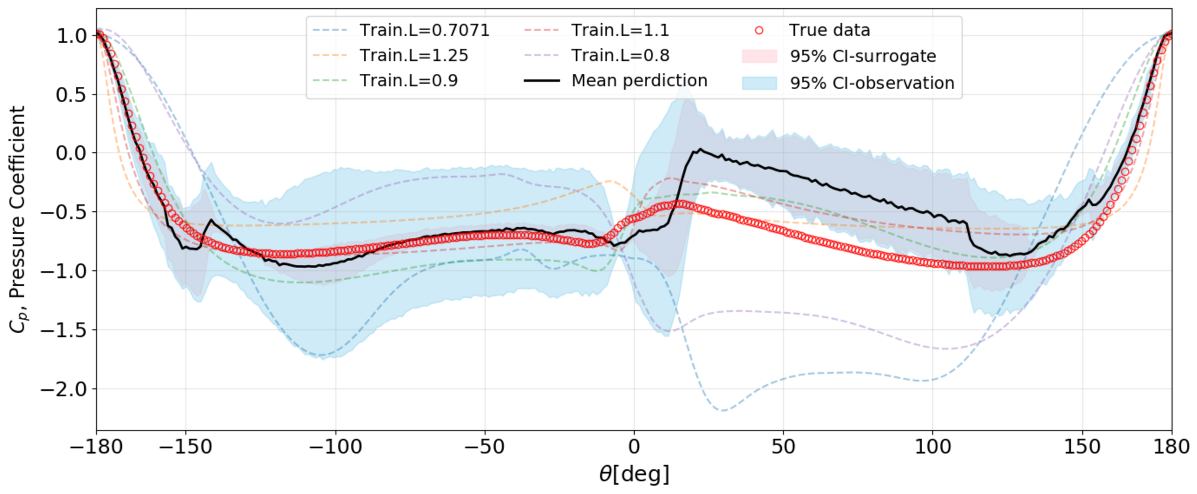


Figure 25: Distribution of the pressure coefficient over the ellipse boundary. The diameter of the ellipse is assumed to be uncertain. Using a set of training simulations, the pressure coefficient for an unseen geometry is predicted and compared to the true simulated data.

A new feature of UQit is a set of tools for estimating uncertainty in time-averaged quantities of turbulence simulations. The reliable estimation of such uncertainties is crucial considering the limitation in computational resources especially for wall-bounded turbulent flows at moderate to high Reynolds numbers that are relevant to engineering applications. The uncertainty estimators are diverse and include batch-based methods as well as methods which work based on modelling the autocorrelation function of time series. We have recently developed a novel methodology to estimate time-averaging uncertainty in high-order and non-linear turbulence statistics. The method has been validated and applied to direct numerical simulations of wall-bounded turbulent flows. As a key feature, the proposed method accurately accounts for all relevant autocorrelation and cross-covariances between the turbulence signals and associated sample-mean estimators. As another main achievement, through a collaboration between KTH and Fraunhofer SCAI, we have successfully developed a framework for in-situ estimation of uncertainties due to time-averaging [32]. The uncertainty estimator is based on modelling the autocorrelation function of time samples, where a novel low-storage updating approach for such models is introduced. Flow solver and UQit are linked through a VTK-based interface. The framework is flexible and independent of the flow solver, and also shows an excellent scaling while adding very small computational overhead. For large-scale simulations of turbulence, this technology will provide a reliable tool for online monitoring of the convergence of turbulence statistics.

In another collaboration between KTH and Fraunhofer SCAI, 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 diameter of the ellipse was 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 26. 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.

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 half of EXCELLERAT to efficiently perform modal decompositions in parallel on large data volumes, in the second half, 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*. Based on the flexible modular reader interface of the decomposition tool, a new parallel reader has been implemented based 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 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 26 (a). The instantaneous flow field of the turbulent flame is visualized by an iso-surface of the Q-criterion coloured by the magnitude of the velocity vector. The DMD analysis is performed on the three-dimensional velocity and pressure field using 2000 snapshots. The resulting spectrum of the DMD analysis, showing the amplitude of each mode as a function of the dimensionless frequency is given in Figure 26 (b). One dominant mode is clearly visible, marked by a red dot at Sr=0.55 matching the dimensionless frequency of the PVC. 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 26 (c). It shows the iso-surface of the Q-criterion coloured by the radial velocity component.

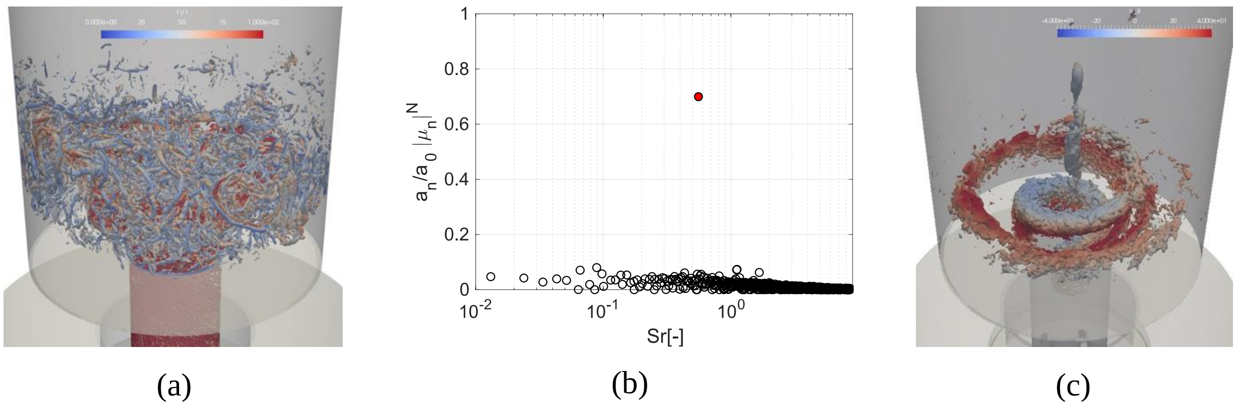


Figure 26: DMD analysis performed on the flow field of a turbulent flame. (a) Instantaneous flow field, (b) Spectrum of the DMD, (c) Reconstruction of the dominant DMD-mode.

Since the modal decomposition toolkit developed in EXCELLERAT addresses a broad user community having various backgrounds and expertise in HPC applications, a user-friendly implementation allowing an efficient workflow is mandatory. To achieve this requirement, an implementation with a client/server architecture has been realized, which is explained in Figure 27. With this implementation, the actual parallelized modal decomposition is performed on the server, i.e., the HPC cluster, which is connected via TCP with the client, i.e., the graphical user interface (running on the local desktop computer. 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 client/server configuration via Catalyst enabling in-situ visualization. In addition to the in-situ visualization using Catalyst, an additional interface using SENSEI was implemented in the last year of EXCELLERAT. This was done to enable a connection to the visualization software Vistle developed in Task 4.1, which utilizes the SENSEI interface (see Section 3.1.2). With this implementation, the modal decomposition toolkit is also able to perform in-situ visualization with the Vistle software.

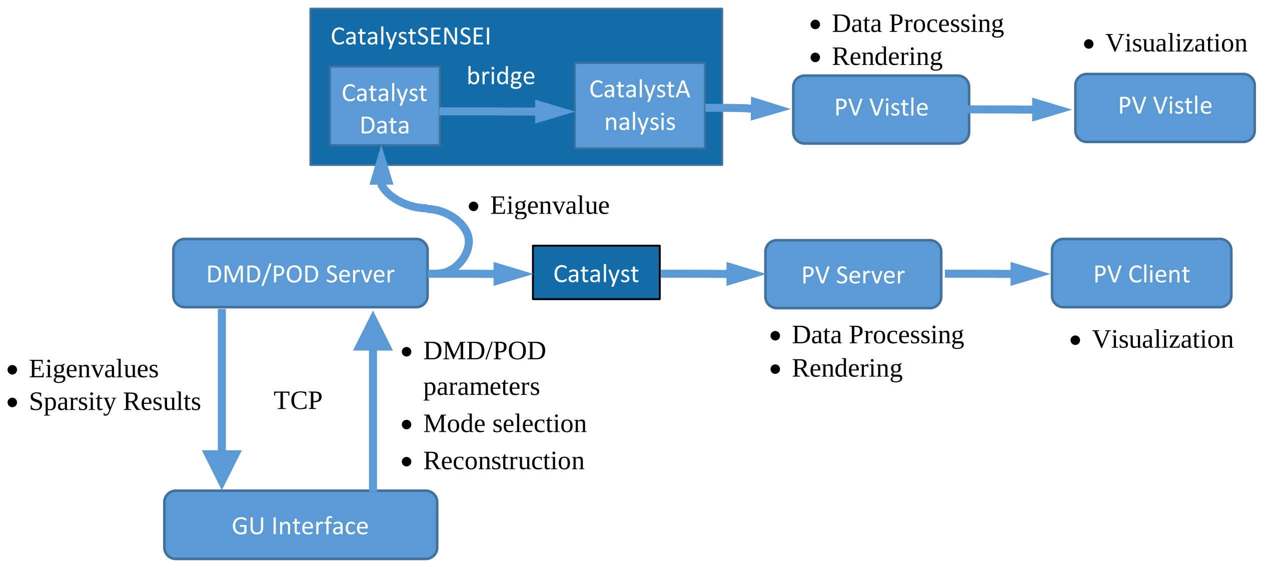


Figure 27: Client/Server structure of the EXCELLERAT modal decomposition toolkit.

To enable an efficient and interactive workflow with the modal decomposition toolkit implemented in a user-friendly client-server architecture, it is essential that the runtime of the decomposition is sufficiently small. For time-resolved large-scale simulations, the snapshot matrix which has to be decomposed reaches a size of up to O(10^9) times O(10^3) matrix entries. Such simulations typically run on pre-Exascale HPC systems. The computationally most expensive part of the modal decomposition is a singular value decomposition (SVD) of this very large, tall (“skinny”) snapshot matrix. To reduce the overall computational time of the decomposition, consequently increasing the usability of the toolkit, various SVD algorithms have been implemented and tested in the last year of EXCELLERAT. Initially, the decomposition toolkit used the parallel SVD routine (pdgesvd) implemented in the ScaLAPACK library. Since the library is based on a two-dimensional block-cyclic distribution, this routine uses intensive communication between ranks and therefore does not perform or scale well for skinny matrices which typically occur in HPC CFD applications. To circumvent this problem and achieve a good parallel performance for exascale applications, two alternative algorithms were implemented in the toolkit: SVD using the tall/skinny QR decomposition TSQR decomposition, and SVD using an eigenvalue decomposition (EVD) of the cross-product matrix. Here is the main idea of both:.

SVD using TS<u>QR</u> decomposition:

$$\begin{aligned} A &= QR \\ &= Q\,(U_R \Sigma_R V_R^T) \\ &= (Q\,U_R)\Sigma_R V_R^T \\ A &= \quad U \quad \Sigma\ \mathrm{V}^{\mathrm{T}} \end{aligned}$$

SVD using EVD of the cross-product matrix ($\boldsymbol{A^T A = V \Lambda V^T}$):

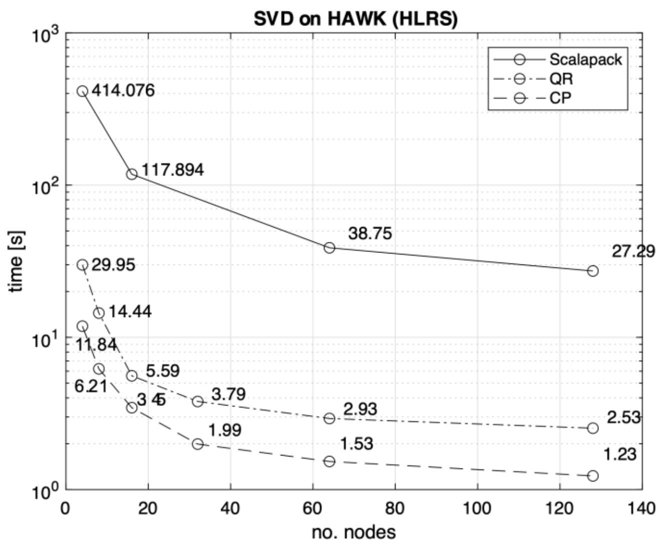
$A^T A = \ (V\Sigma U^T)(U\Sigma V^T) = V\Sigma^2 V^T$

→ eigenvalues Λ = singular values Σ^2

→ eigenvectors V= right singular vectors V

Left singular vectors $U = AV\Sigma^{-1}$

Using the alternative algorithms, the communication was substantially reduced leading to a significant reduction of the computational time as shown in Figure 28. After introducing the alternative SVD algorithms into the decomposition framework, the toolkit is now ready to process the results of large, exascale computations.



HAWK:
2 x AMD EPYC 7742 per node
- 64 cores @ 2.25 GHz per CPU
- 256 GB DRAM per node

Figure 28: SVD using ScaLAPACK, the TSQR decomposition (QR) and the EVD of the cross-product matrix (CP)

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 in Section 5.6. 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 was 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 in Section 5.6.

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.

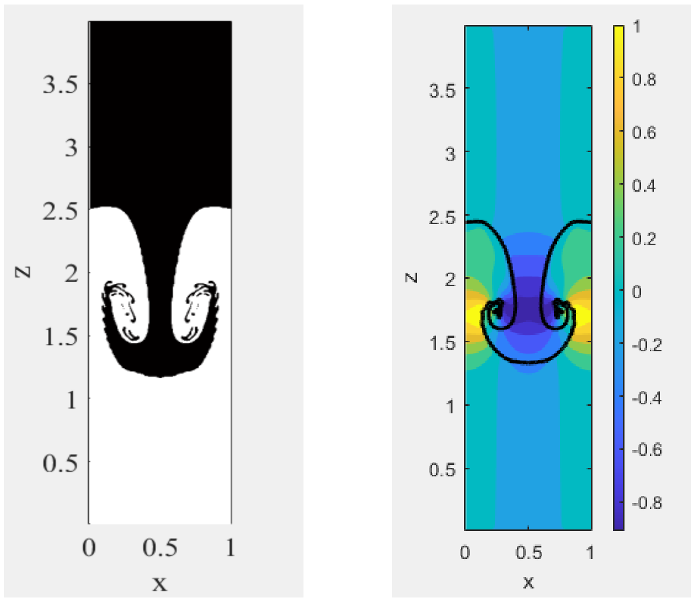


Figure 29: Two images created by the two Matlab post-processing routines.

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 Fortran90 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. 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 Section 5.6. 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 that 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 (see D5.7) 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 the journal papers [30] and [26]. 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. 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, introduced to prepare for exascale platforms.

5 Task 4.4: Data management

5.1 Introduction

The Data Exchange & Workflow Portal in form of a software tool combines the topics data transfer, data management, data dispatching and data scheduling. Therefore, the platform merges the effort of Task 3.6 “Data dispatching through data transfer” and Task 4.4 “Data Management” which cannot be separated.

A typical data round trip for an exascale simulation workflow is shown in Figure 30 - pointing out several challenges. The round trip describes the overall process which we are trying to adapt to with the help of a first proof of concepts.

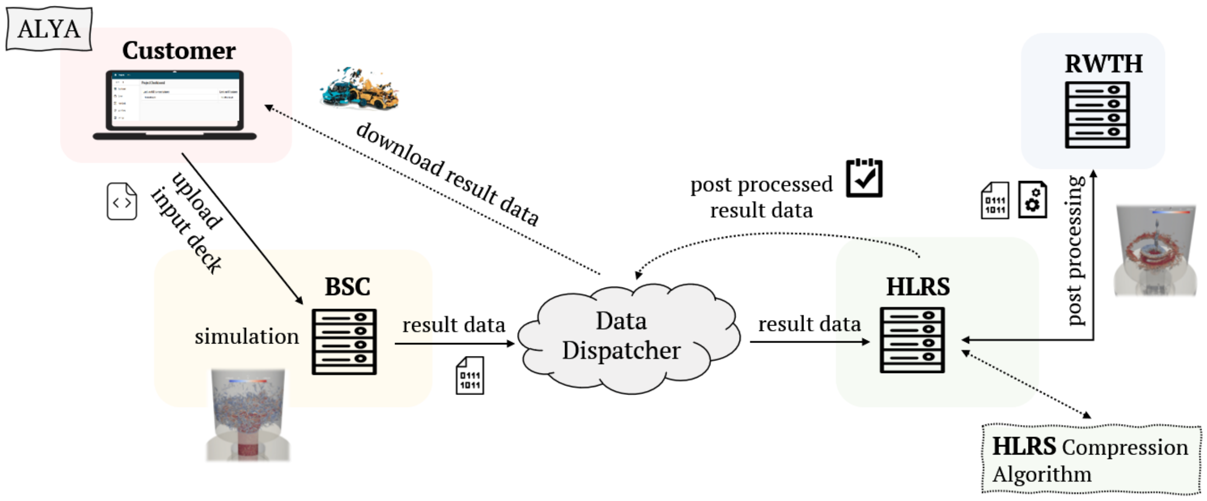


Figure 30: Data Roundtrip.

The goal is to complete a data transfer round trip with one of the EXCELLERAT use cases which 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 is optimized, but it is important to consider that there is a specific lifecycle around it that is particularly important for the end user.

5.2 Challenges regarding exascale

Companies face various problems while dealing with HPC computations, HPC in general or even access to HPC resources. Small companies in particular cannot meet certain requirements because their existing internet connection has not a sufficient bandwidth or Quality-of-Service for the necessary HPC connection setup.

When needing high-fidelity simulations, results usually contain large amounts of data. As transfer bandwidth and storage are restricted, careful consideration must be given to what data is sent back to the client. Altogether, data transfer must be efficient and secure.

We address these challenges in several tasks of EXCELLERAT. This is described in detail in the following section.

5.3 Solution

Through the Data Exchange & Workflow Portal, which is developed within the Tasks 3.6 and 4.4 by SSC, the challenges described in the previous section can be addressed. 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 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 Data Exchange & Workflow 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 developed 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 are executed on the same architecture as the main simulation. Besides providing better physical insights, this reduces 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 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.

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

5.4 Introduction of the platform interface

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 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.

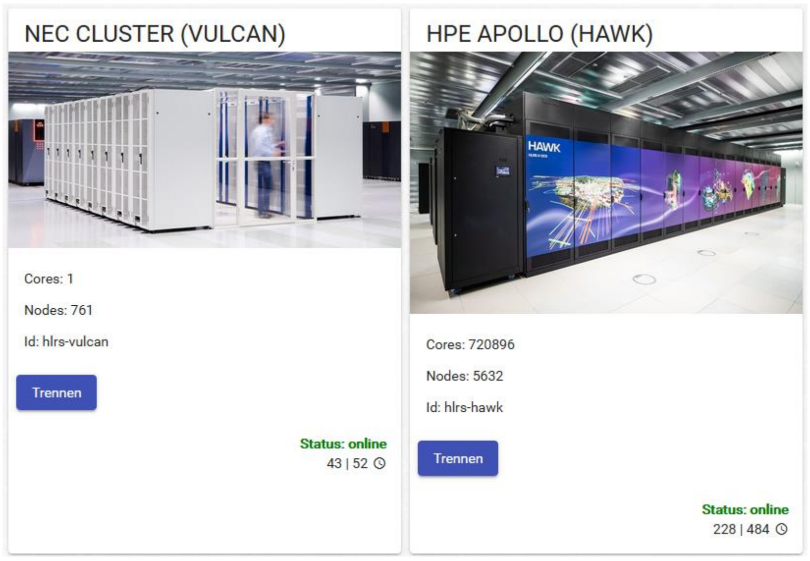
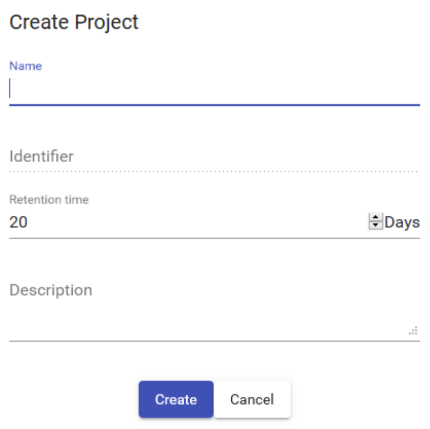


Figure 31: Connection to an HPC cluster.

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 should be (1-30 days with a possible extension of three times, which corresponds to a total of 120 days). A description is optional.


Create Project

Name

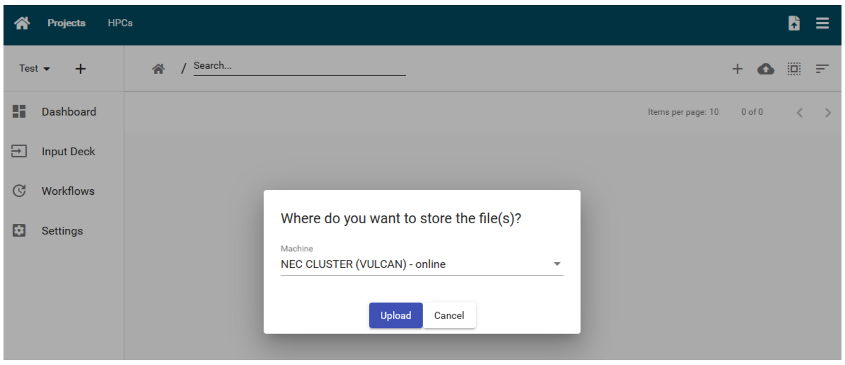
Identifier

Retention time
20 Days

Description

Figure 32: 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 HPC centres with several machines can 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.

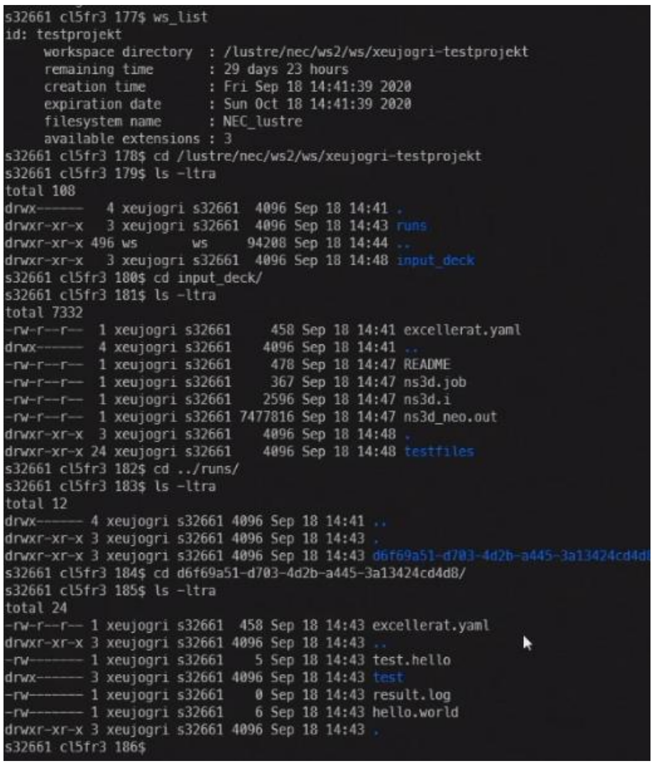


Where do you want to store the file(s)?

Machine
NEC CLUSTER (VULCAN) - online

Figure 33: Uploading an input deck.

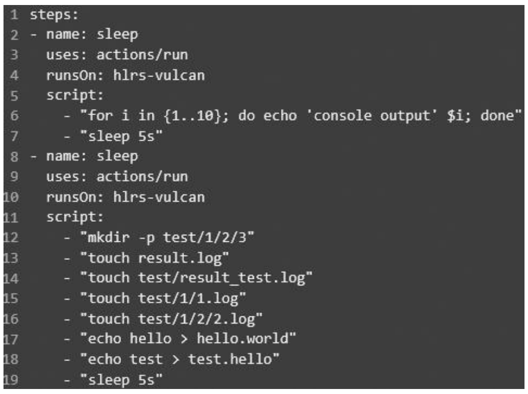
All uploaded files end up in the editor's workspace. For the platform, a specific folder structure is created on the HPC cluster so that all files can be found again. However, all modifications that take place outside the platform are not monitored, since no direct accesses are integrated into the cluster.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



```
s32661 cl5fr3 177$ ws_list
id: testprojekt
     workspace directory  : /lustre/nec/ws2/ws/xeujogri-testprojekt
     remaining time       : 29 days 23 hours
     creation time        : Fri Sep 18 14:41:39 2020
     expiration date      : Sun Oct 18 14:41:39 2020
     filesystem name      : NEC_lustre
     available extensions : 3
s32661 cl5fr3 178$ cd /lustre/nec/ws2/ws/xeujogri-testprojekt
s32661 cl5fr3 179$ ls -ltra
total 108
drwx------   4 xeujogri s32661  4096 Sep 18 14:41 .
drwxr-xr-x   3 xeujogri s32661  4096 Sep 18 14:43 runs
drwxr-xr-x 496 ws       ws     94208 Sep 18 14:44 ..
drwxr-xr-x   3 xeujogri s32661  4096 Sep 18 14:48 input_deck
s32661 cl5fr3 180$ cd input_deck/
s32661 cl5fr3 181$ ls -ltra
total 7332
-rw-r--r--  1 xeujogri s32661     458 Sep 18 14:41 excellerat.yaml
drwx------  4 xeujogri s32661    4096 Sep 18 14:41 ..
-rw-r--r--  1 xeujogri s32661     478 Sep 18 14:47 README
-rw-r--r--  1 xeujogri s32661     367 Sep 18 14:47 ns3d.job
-rw-r--r--  1 xeujogri s32661    2596 Sep 18 14:47 ns3d.i
-rw-r--r--  1 xeujogri s32661 7477816 Sep 18 14:47 ns3d_neo.out
drwxr-xr-x  3 xeujogri s32661    4096 Sep 18 14:48 .
drwxr-xr-x 24 xeujogri s32661    4096 Sep 18 14:48 testfiles
s32661 cl5fr3 182$ cd ../runs/
s32661 cl5fr3 183$ ls -ltra
total 12
drwx------ 4 xeujogri s32661 4096 Sep 18 14:41 ..
drwxr-xr-x 3 xeujogri s32661 4096 Sep 18 14:43 .
drwxr-xr-x 3 xeujogri s32661 4096 Sep 18 14:43 d6f69a51-d703-4d2b-a445-3a13424cd4d
s32661 cl5fr3 184$ cd d6f69a51-d703-4d2b-a445-3a13424cd4d8/
s32661 cl5fr3 185$ ls -ltra
total 24
-rw-r--r-- 1 xeujogri s32661  458 Sep 18 14:43 excellerat.yaml
drwxr-xr-x 3 xeujogri s32661 4096 Sep 18 14:43 ..
-rw------- 1 xeujogri s32661    5 Sep 18 14:43 test.hello
drwx------ 3 xeujogri s32661 4096 Sep 18 14:43 test
-rw------- 1 xeujogri s32661    0 Sep 18 14:43 result.log
-rw------- 1 xeujogri s32661    6 Sep 18 14:43 hello.world
drwxr-xr-x 3 xeujogri s32661 4096 Sep 18 14:43 .
s32661 cl5fr3 186$
```

Figure 34: Workspace structure on an HPC filesystem.

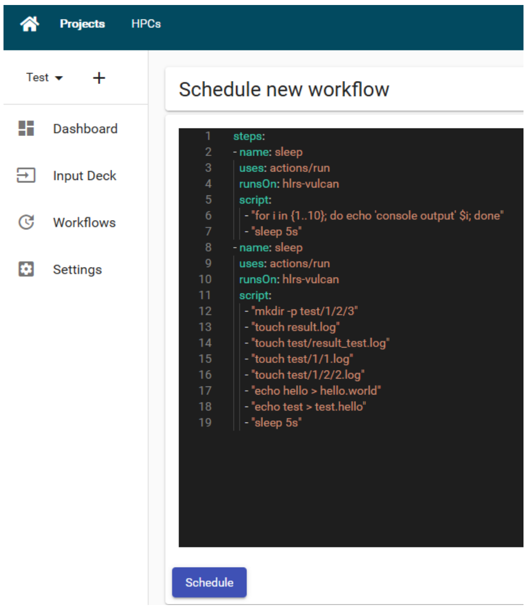
So far you can specify an “excellerat.yaml” file for each project. This control file describes the workflows and how the simulation should look like and what should be done with it in the workspace. In there, you can also specify scripts that may run in pre- and post-processing. These could be included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



```
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 35: Example of an excellerat.yaml file

After creating a new workflow, the uploaded excellerat.yaml file 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 and prepares the run script ("run.sh") from the input files and passes it to the simulation.



```
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 36: 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 notifying 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.

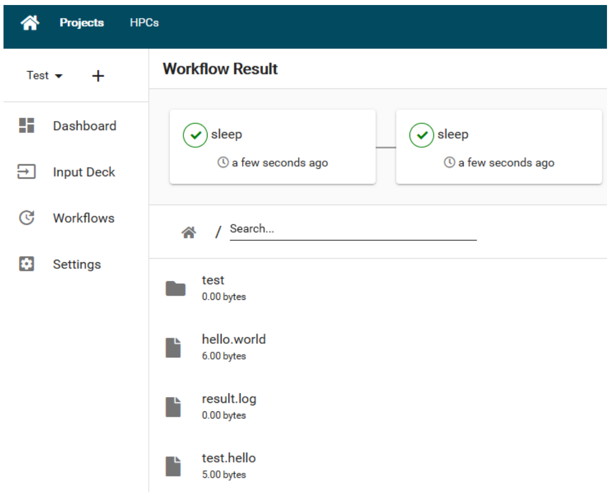


Figure 37: Workflow overview.

5.5 TPLS I/O for exascale platform

In the framework of the same tasks, UEDIN developed 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. This is following closely the definition of Task 4.4 and particularly includes 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.

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 have been introduced. First, ASCII data formats were replaced with binary HDF5 files. Second the end user is enabled to specify which data to read/write instead of reading/writing unwanted data. Finally, cell coordinates have been removed from data files, as they can be reconstructed 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 at the beginning HDF5 files were 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 HDF5 binary file. Novel memory and I/O hierarchies will be supported by the PETSc I/O library, as PETSc releases new versions for new hardware.

5.6 Development Progress

Since the last reporting period, 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 the users 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 final step, which is not yet fully implemented, is to compress the files on the way back.

Furthermore, many minor features have been added since the last reporting period. It is now possible to connect more than one HPC machine to the platform. Users can connect directly to the HPE Apollo (Hawk) and the NEC cluster (Vulcan) of 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 which state the executed job is in.

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 (e.g. Hydrograv and Festo) and incorporating their user feedback into the platform. The knowledge gained from this can be incorporated into the development of the platform.

5.7 Summary of Data Management Activities

The main focus of the work in Task 4.4 *Data Management* and Task 3.6 *Data Transfer* has been to put on the development of the Data Exchange & Workflow Portal platform and to complete the data roundtrip for exascale engineering simulations. The most important outcomes are listed below:

- Providing access to the most relevant HPC processes to the users (upload Input Deck, run simulation, download results)
- Secure, traceable, automated transfer between data generators and HPC centres
- Various changes in the platform's user interface to improve usability
- Within the WP-transversal Data Transfer and Data Management Working Group Task 3.6 and 4.4, we could contribute to the EXCELLERAT service portfolio. For example, the platform was presented in 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.

Beside the further development of the software, a piloting study was conducted to evaluate the usability of the tool for medium sized-enterprises. The access to and usage of HPC for industrial partners was a key requirement and goal for the developed platform. In collaboration with ten industrial partners (e.g. Hydrograv, Festo, Eberspächer, Elringklinger, Syntegon) the platform was tested with real data and the use cases of the partners were examined to see if the tool can meet their needs.

In general, most of the users were very satisfied with the solution. Especially the dashboard with the traceability of jobs that have been executed in a user-friendly interface were pointed out as an improvement to the currently existing command-line-only solutions. Some of the partners used the platform on a regular basis during the whole piloting phase.

The feedback regarding additional features and improvements was analysed and the most important issues were scheduled for the further development. Those include:

- Reusable templates for workflows

  Many times, users will have multiple calculations with just slightly changed input data to simulate the impact of changes. Therefore, the possibility to create a reusable workflow will save time and make it easier to secure a reliable constant test environment.

  The implementation will include saving a defined workflow as a template for future workflow executions. Additionally, templates can be used by multiple users (e.g. colleagues of the same company or team).
- Two-Factor-Authentication

  To further strengthen security, a two-factor authentication will be setup. Possible technical solutions will be evaluated and implemented.
- Loading of environment variables
  - Environment variables play an important role in simulations. Besides changing input data, this is the most common use case to understand the effect of a changed environment. Supporting the use of variables will increase the usability and add to the benefit for users.

In addition to the mentioned new features, next steps include a local installation of the platform within the HLRS infrastructure as an on-premise solution and the rollout for production use with a licensing model for commercial use.

6 Task 4.5: Usability

The general goal of the usability task is to provide workflows and best practices for an engineering simulation's entire life cycle, from pre-processing, including modelling and meshing, and 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.

Throughout the duration of the project, the usability task has taken an observatory role, monitoring the developments in the core codes and gathering experiences from use-case owners after their runs, from prototype runs to final demonstration runs. A common theme among several core codes is the efficient use of emerging technologies and accelerators, particularly towards the end of the project. Until recently, most, if not all, accelerator-related efforts within EXCELLERAT 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 announced EuroHPC pre-exascale systems [33] and announced DoE exascale systems [34], support and knowledge about OpenMP 5 (offloading) or HIP [35] might be equally important for developers of engineering simulation codes. Unfortunately, due to delayed installation of these EuroHPC systems, and a worldwide supply shortage of AMD accelerator technology, the centre could not gain significant experience with these technologies during the project's lifetime.

Gathered experiences from executing the use-cases and the efficient use of accelerators have been reported in a separate centre-wide (WP2, WP3, WP4) best practice guide for exascale engineering simulations. This combined guide also contains gathered data and post-processing experiences from the more data-intensive runs in the final phase of the project.

7 Conclusion

To conclude, the development of enhanced services to support exascale engineering simulation workflows has progressed as planned within work package 4. The main achievement during the project has been to formulate and validate prototypes of the enhanced services outlined in deliverable D4.2. Co-design activities were established with a strong connection to the work performed in work package 3. Scalable in-situ visualization workflows for both interactive and non-interactive analysis have been developed and tested with several EXCELLERAT core-codes. Similarly, in-situ data analytics and uncertainty quantification frameworks have been 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 an input deck to downloading post-processed results. These services have been designed and implemented based on use-cases' needs, either using derived model problems or scaled-down formulations of the full use-cases. In the final phase of the project, developed services have advanced past the prototype stage and have been integrated and validated within 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, [Online]. Available: https://www.compbiomed.eu.

[3] HiDALGO, “https://hidalgo-project.eu,” [Online].

[4] ChEESE, “https://cheese-coe.eu/,” [Online].

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

[6] E. Chow, “A Priori Sparsity Patterns for Parallel Sparse Approximate Inverse Preconditioners,” *SIAM Journal on Scientific Computing,* vol. 21, no. 5, pp. 1804-1822, 2000.

[7] E. Chow, “Parallel Implementation and Practical Use of Sparse Approximate Inverse Preconditioners With a Priori Sparsity Patterns,” *International Journal of High Performance Computing,* vol. 15, no. 5, 2001.

[8] T. A. Davis and Y. Hu, “The University of Florida Sparse Matrix Collection,” *ACM Trans. Math. Softw.* , vol. 38, no. 1, 2011.

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

[10] M. Karp, N. Jansson, A. Podobas, P. Schlatter and S. Markidis, “Optimization of Tensor-product Operations in Nekbone on GPUs,” in *Poster presented at the International Conference for High Performance Computing, Networking, Storage and Analysis (SC’20 Poster),* 2020.

[11] LAPACK, “http://www.netlib.org/lapack/,” [Online].

[12] ScaLAPACK, “http://www.netlib.org/scalapack/,” [Online].

[13] SLATE, “https://icl.utk.edu/slate/,” [Online].

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

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

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

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

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

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

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

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

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

[23] 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.

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

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

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

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

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

[29] C. Gscheidle and J. Garcke, “Explorative In-situ Analysis of Turbulent Flow Data Based on a Data-Driven Approach,” in *ECCOMAS 2022,* Oslo, 2022.

[30] S. Rezaeiravesh, R. Vinuesa and P. Schlatter, “An uncertainty-quantification framework for assessing accuracy, sensitivity, and robustness in computational fluid dynamics.,” *Journal of Computational Science, Article-in-press,* 2022.

[31] S. Rezaeiravesh, R. Vinuesa and P. Schlatter, “On Numerical Uncertainties in Scale-Resolving Simulations of Canonical Wall Turbulence,” *Computer & Fluids,* vol. 227, pp. 1-21, 2021.

[32] C. Gscheidle, S. Rezaeiravesh, J. Garcke and P. Schlatter, “In-situ estimation of time-averaging uncertainties in turbulent flow simulations,” in *Workshop MMCP, HPC Asia 2022,* Kobe, Japan, 2022.

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

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

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

[36] 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.

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

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 [36].

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 libraries (further referred to as "the adapter") implementing the same interface as VisIt but with Vistle's own implementation.

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 38 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. Custom commands that take string input are respectively represented as string input fields that trigger the corresponding command on change. When the connected simulation produces data, depending on the configured "frequency" parameter and the connected output ports of the module, data objects are transformed and copied to SHM in the adapter and then passed to the Vistle pipeline. This will trigger an automatic execution of the LibSimController module and therefore propagates the execution trough the following connected modules. Depending on the "keep timesteps" parameter a timeline or a single snapshot is generated.

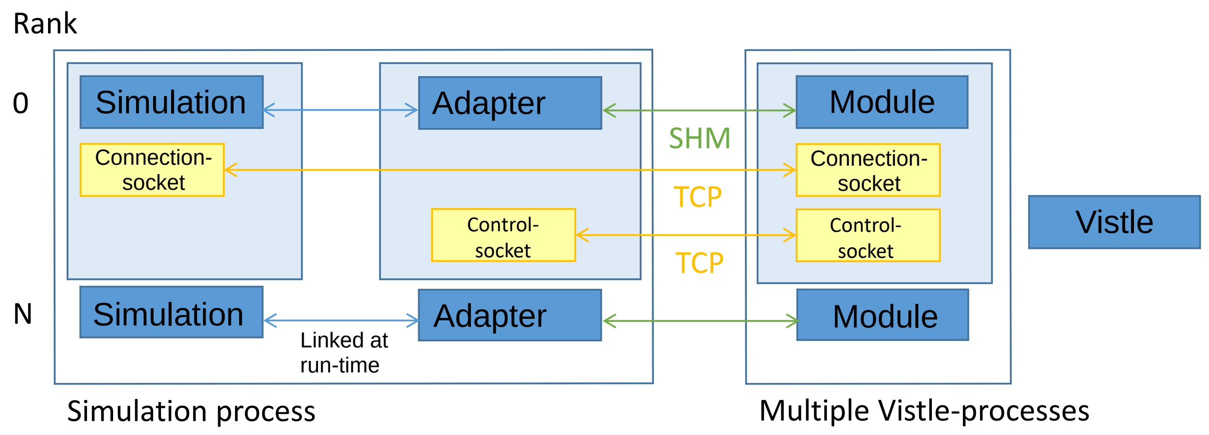


Figure 38: Schematical data exchange between a LibSim-instrumented simulation and Vistle in multi-process mode

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 39 shows how data is exchanged with this architecture.

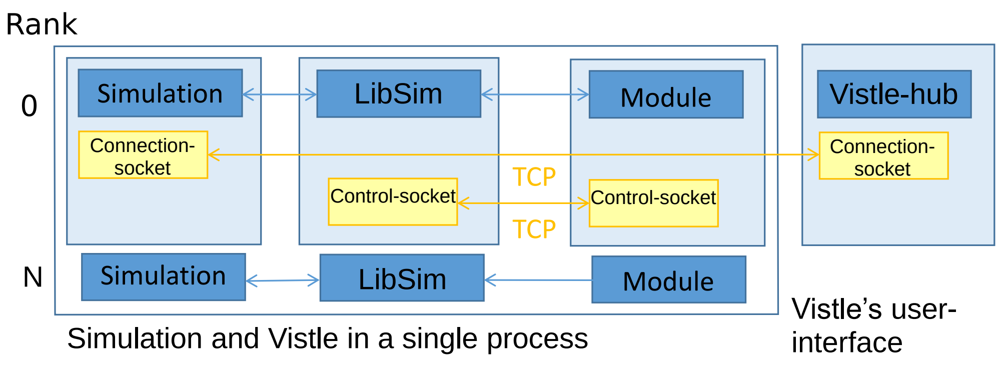


Figure 39: Schematical data exchange between a LibSim-instrumented simulation and Vistle in single-process mode

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. This MPI communication is non-busy to not block other simulation/Vistle.threads. 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 16 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 is a compile time decision. Because of the compile time linking the only specific environment variable is VISTLE_KEY that has to be set if a remote connection with the compute cluster is desired. 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 SENSEI MPIManager will automatically take care of this when used by the simulation. The pipeline can then be controlled by attaching 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. For the default file location this is done automatically. 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_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