H2020-INFRAEDI-2018-2020

The European Centre of Excellence for Engineering Applications

Project Number: 823691

D4.5

Best Practices and Tools for Visualization, Data Management and Analytics of Engineering Simulations at Exascale

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

Workpackage:	D4.5	Best Practices and Tools for Visualization, Data Management and Analytics of Engineering Simulations at Exascale	
Author(s):	Niclas Jansson		KTH
	Dennis Grieger		USTUT
	Christian Gscheidle		Fraunhofer
	Saleh Rezaeiravesh		KTH
	Jens Gerle		SSC
	Patrick Vogler		USTUTT
Approved by	Executive Centre Management		2022-06-14
Reviewer	Karin Schlottke		USTUTT
Reviewer	Michael Wagner		DLR
Dissemination Level	PU		

Date	Author	Comments	Version	Status
2022-05-30	Niclas Jansson	Initial draft	V0.0	Draft
2022-05-31	Karin Schlottke	Review 1	V0.1	
2022-06-02	Michael Wagner	Review 2	V0.2	
2022-06-03	Patrick Vogler	Incorporation of the necessary changes	V1.0	Final Version

Executive Summary

This document summarises the developments and achievements in Work Package 4 and how these contributed to the combined Best Practices Guide (BPG) of the EXCELLERAT project. Work Package 4 deals with the development of EXCELLERAT's enhanced services: co-design, visualization, data analytics and data management. In essence, developing tools for an application's entire life cycle. Although the work package covers several aspects of a simulation life cycle, this document only highlights the contributions related to visualization, data analytics and data management.

The experience from developing scalable in-situ visualization workflows for EXCELLERAT's use-cases within the visualization task (Task 4.2) has formed the basis for the recommendations and tools – both remote and in-situ visualization techniques – for (pre-) exascale engineering simulations in the post-processing section of the BPG. Whereas, not only commonly used tools are discussed but also tools like Vistle, which was further developed and enhanced during the EXCELLERAT project.

Data analytics (Task 4.3) has contributed their experience and new in-situ developments to address the post-processing bottleneck when computing statistics from the large amount of data from post-petascale turbulence simulations. Furthermore, the development of efficient and scalable uncertainty quantification methods within the data analytics task is the foundation for the survey and recommendations on uncertainty quantification methods in the simulation workflow and result feedback section, mainly focusing on UQit, the open-source uncertainty quantification toolkit developed within EXCELLERAT.

Developing best practices and support for managing the large amount of data produced and used in engineering applications has been the focus of Task 4.3: data management. Experiences gained from further developing tools (e.g. a data management platform) and methods (e.g. compression algorithms) to support the complete data round trip, from transferring input decks to an HPC system, moving simulation results to post-processing resources and downloading results back to the end-users, both securely and efficiently, have contributed to the data transfer sections in the pre- and post-processing parts of the BPG.

The development of enhanced services to support exascale engineering simulation workflows in Work Package 4 has, across the work package's tasks, been one of the main contributors to the BPG of EXCELLERAT. For the tasks not covered by this document, please refer to D4.7 "*Final Report on Enhanced Services Progress"* for more details on how the knowledge and developments in these tasks have, directly and indirectly, contributed to the BPG.

ANNEX

Best Practice Guide

List of abbreviations

AMR	*Adaptive Mesh Refinement*
AoS	*Array of Structs*
CFD	*Computational Fluid Dynamics*
CFL	*Courant–Friedrichs–Lewy condition*
CLI	*Command Line Interface*
CoE	*Centre of Excellence*
CPU	*Central Processing Unit*
CUDA	*Compute Unified Device Architecture*
DLB	*Dynamic Load Balancing*
DMA	*Direct Memory Access*
DMZ	*Demilitarised Zone*
FPGA	*Field-Programmable Gate Array*
GASPI	*Global Address Space Programming Interface*
GMRES	*Generalized Minimal Residual*
GPU	*Graphics Processing Unit*
HBM	*High Bandwidth Memory*
HDL	*Hardware Description Language*
HLS	*High Level Synthesis*
HPC	*High-Performance Computing*
HPDA	*High Performance Data Analytics*
ISA	*Instruction Set Architecture*
MPI	*Message Passing Interface*
PE	*Parallel efficiency*
PETSc	*Portable, Extensible Toolkit for Scientific Computation*
PRACE	*Partnership for Advanced Computing in Europe*
RCB	*Recursive Coordinate Bisection*
SEM	*Spectral Element Method*
SFC	*Space Filling Curve*
shm	*shared memory*
SoA	*Struct of Arrays*
Spliss	*Sparse Linear Systems Solver*
SpMV	*Sparse matrix–vector multiplication*
TDP	*Thermal Design Power*
UDF	*User Defined Function*
UPC	*Unified Parallel C*
VE	*Vector Engines*
VPN	*Virtual Private Network*

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This document is an outcome of EXCELLERAT the European Centre of Excellence for Engineering Applications. The centre of excellence (CoE) was established in December 2018 as a project that was funded in the Horizon 2020 framework programme to work towards a sustainable foundation for a central European knowledge and competence hub for all stakeholders participating in the usage and exploitation of high-performance computing (HPC) and high-performance data analytics (HPDA) in engineering.

Having worked together throughout the 42 months of the initial funding phase, in this document the EXELLERAT consortium presents what were found to be the best practices in the execution of engineering applications on state-of-the-art HPC-systems on the path towards the exascale era.

As all tasks and actions of EXCELLERAT are driven by the requirements that are posed by the usage of HPC for the execution of engineering applications this best practices guide is also oriented along the typical usage path of scientific computing applications in engineering research and development.

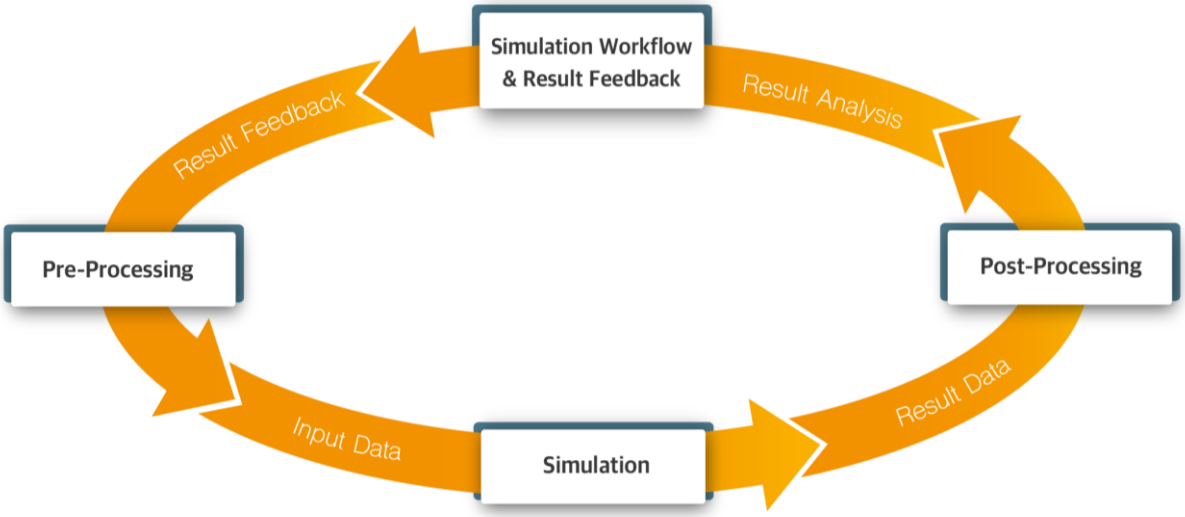

Figure 1: Engineering Workflow

As depicted in Figure 1 the engineering workflow is composed of four steps. The steps are derived from the classical simulation task in engineering in which a given physical problem is described via a set of partial differential equations whose solution is being made universally computable by the application of a discretization scheme like e.g. the Finite Volume Method.

In order to be able to execute the implementation of the discretized model, the pre-processing step, in which the input data of the model are prepared, is the first task. After the execution, i.e. the simulation step, the produced result data have to be analysed in the Post-Processing step. In the Result Feedback step, conclusions are drawn from the analyses of the results data. These are used to modify the input data and thus trigger the next design cycle.

If one takes a closer look to the usage of HPC in engineering, all arising tasks can be assigned to one of these four steps even in applications of pure HPDA in which the simulation step not necessarily is an actual simulation model but rather the execution of an analysis software.

During the first phase of EXCELLERAT, the tasks that are most essential and that require the most attention for the successful and efficient use of HPC in engineering, especially when targeting exascale applications, were identified on the basis of various use cases. The approaches and "How To" solutions that were found by the EXCELLERAT consortium to be taken best when trying to approach exascale applications in engineering are presented in the following chapters of this "Best Practices Guide".

- **Chapter 2 - Pre-Processing:**

  In this step, the focus was on efficient data transfer between the user site and the HPC centre and on the efficient creation of baseline meshes.

- **Chapter 3 - Simulation**

  Since the core component of all considered use cases was a large-scale simulation application, most of the efforts spent, went into the simulation step. One bottle neck for the successful approach towards exascale applications in engineering that was identified even before EXCELLERAT started was the necessity of efficient methods for adaptive mesh refinement. In addition, best practices for the related tasks domain decomposition and load balancing are presented along with results in the area of discretization methods, validation as well as parallel input and output (I/O).

- **Chapter 4 - Post-Processing**

  When it comes to the Post-Processing step for large scale engineering applications, the classical approach of "Writing data to disk, read back and visualize" is no longer possible since the data sets are getting too large and complex to be properly analysed by purely "looking at it". Due to that, best practices for data reduction and compression, data analytics i.e. HPDA, remote and in-situ visualization as well as efficient transfer of the analyses results' back to the user's site are presented.

- **Chapter 5 - Simulation Workflow & Result Feedback**

  To enable efficient and fast design processes, automatic methods for result feedback are another key component to harvest the possibilities of HPC in engineering. This is especially true when it comes to uncertainty quantification methods.

- **Chapter 6 - Workflow Overarching Activities**

  In addition to the tasks that can directly be assigned to the individual steps, performance engineering and efficient implementation play an essential role in the use of HPC. Since this is true for all applications executed on large scale HPC resources independent of their position within the engineering workflow, theses general activities are grouped together in chapter 6 - Workflow Overarching Activities.

2 Pre-Processing

In the engineering workflow the pre-processing step collects not only all tasks concerned with model preparation like domain discretizations i.e. meshing, physics specification and boundary application etc. but also all tasks connected with data transfer from the user's site to the HPC centres. Two major areas in which considerable improvements are needed were identified during the cause of the project.

1.) Data transfer to HPC-Site
When moving from academic usage in which the application developer or even HPC-experts are directly involved in the execution of engineering applications on HPC resources, towards production like usage scenarios in which the user of the application is neither a HPC expert nor an application developer the necessity to ease the usage of HPC systems especially during the preparation of the input and the compute tasks becomes essential and can be of great benefit in the uptake of HPC methodologies by new stakeholders. In addition, data integrity & security throughout the development process are essential for industrial users to ensure short turnaround times and prevent duplicated efforts. In the case of the EXCELLERAT project a solution and approach to circumvent these problems was developed whose overview and further reading details are presented in section 2.1.

2.) Mesh generation for high order methods
In the industrial and productive usage of computer aided engineering and computational fluid dynamics high order methods are only very rarely used. Even though they can deliver results of superior quality, one of the main bottlenecks of these methods is the preparation of suitable computational grids which can be a complex and time-consuming task once realistic geometries should be treated. With Nek5000 a highly scalable and efficient HPC application of the aforementioned type was selected as a core code of EXCELLERAT. In section 2.2 the developments and best practices found for the generation of high order meshes are summarized.

2.1 Data Transfer (to HPC Site)

Besides handling the data traffic of large amounts security is the main issue when communicating with the HPC site. Typically, each customer is connected to the HPC centre manually via a network connection with individual firewall rules for the corresponding IP addresses. This results in significant effort and potential security issues. Using a software tool allows a single-source-of-contact approach where the main focus for security is the communication between the tool and the HPC site. For the consumer the access to the tool is convenient and secured with user credentials (E-Mail and password) that allows an easier access to HPC resources for industrial partners without a long onboarding process.

Figure 2 shows the architecture of the software. The connection of a user to the tool is always encrypted and secured via TLS (Transport Layer Security). Additionally, rate limits prevent DDoS or similar attacks. The backend VM connects to the DMZ of the HPC site (here: HLRS) with a Site-2-Site VPN connection. From there the machines are addressed via a Secure Shell access.

Another point that can be dealt with efficiently by using a central data platform is the reduction of data and the avoidance of data duplication. The goal of data reduction is essentially to avoid data redundancy on a storage system. With the help of this storage technique, only as much information as necessary is to be written to a non-volatile storage medium in order to be able to reconstruct a file without loss. The more duplicates that are removed, the smaller the amount

of data that needs to be stored or transferred. To do this, the files are first broken down into small data blocks (chunks) and given unique checksums, known as hash values. A tracking database containing all checksums serves as the central control instance.

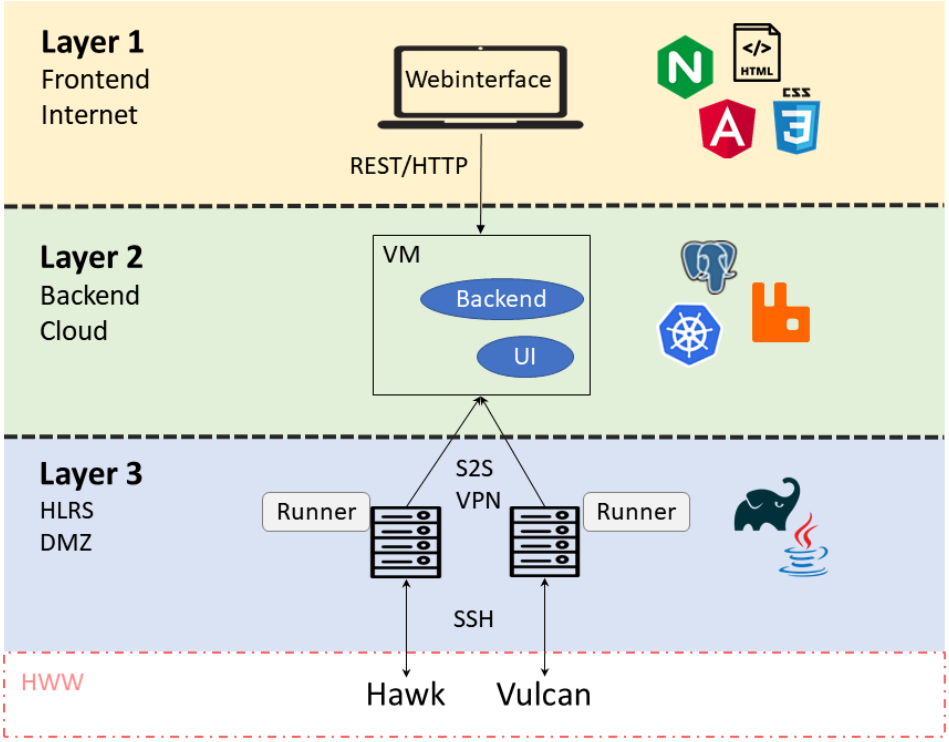

Figure 2: System layout of access and data transfer platform

The data management technique is based, among other things, on the InterPlanetary File System protocol, a peer-to-peer method of storing and sharing hypermedia in a distributed system. Here, the cryptographic hash function SHA256 and Base58 encoding are used. Within the hash trees, the SHA256 checksums are combined, which in turn are combined into a new hash. Therefore, the probability of a possible SHA256 hash collision is very low.

Thus, the identification of redundant chunks is based on the assumption that data blocks with identical hash values contain identical information. In order to filter out redundant chunks, the deduplication algorithm requires newly determined hash values to be matched against the tracking database only. If the deduplication algorithm finds identical checksums, the redundant chunks are replaced by a pointer that points to the location of the identical data block. Such a pointer takes up much less space than the data block itself. The more chunks in a file that can be replaced by placeholders, the less storage space the file requires.

The data reduction takes place in the background of the platform. As soon as a user uploads a file via the platform, a hash tree is calculated from the entire file, which is broken down into 1 MB blocks (chunks). Using this hash tree, the data dispatcher can determine which parts have been uploaded before and which have not. As soon as the data is no longer available in the system, the complete hash tree is deleted.

In conclusion one can state, that with respect to security, using a software as a central access point is a good approach to minimalize efforts and attack points. For users especially from an industrial context the benefit is having a user interface for creating their simulation jobs and a much more convenient access to the HPC site.

2.2 Mesh Generation & Mesh Algorithms

The accuracy and efficiency of the numerical solvers strongly depend on the approximations space on which the solution would be computed, and this translates directly to the generation of the optimal grid for a given use case. However, it is rarely possible to determine such an optimal grid in advance making the meshing process challenging. An important improvement comes with the use of an adaptive mesh refinement (AMR) algorithm allowing for the control of the computation error during the simulation by dynamical adjustment of the computational grid. In addition, AMR may simplify meshing process providing e.g., flexibility of nonconforming mesh, but at the same time makes it more demanding, as efficient AMR requires relatively coarse initial mesh, that still properly represents a domain geometry. This can be a difficult task, especially for Nek5000, as this solver is based on a spectral element method (SEM) and requires high order, hex-based meshes. Building such meshes is an unsolved problem and most of the available meshing software does not support it. In this section we will summarize our experience with hex-mesh generation using an open-source mesher gmsh [1]. It is a natural choice, as Nek5000 provides a robust mesh converter. In the following paragraphs, we will relate to both the scripting language commands and the Fortran interface (mostly in parenthesis).

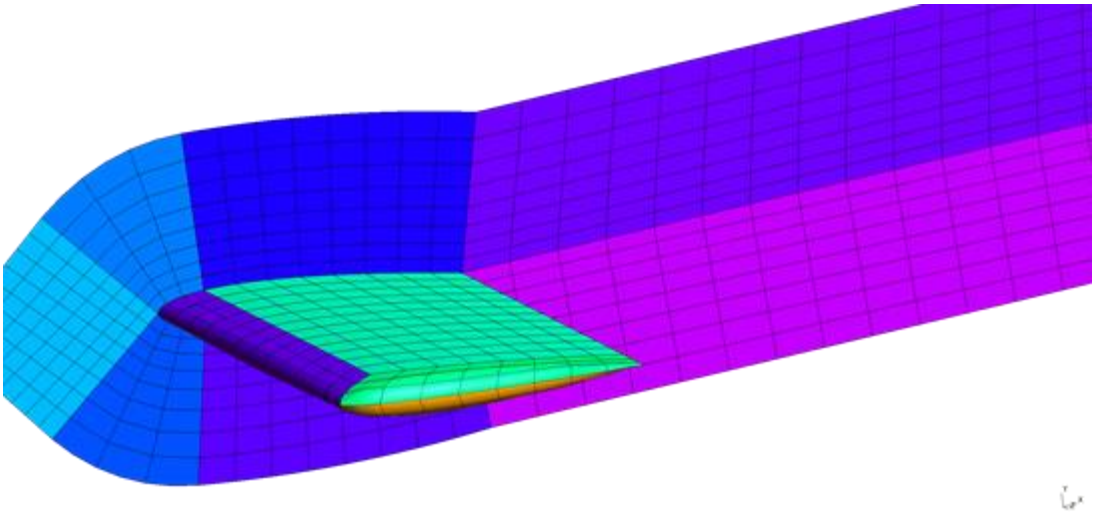

Figure 3: Exemplary split of the computational domain into the set of subsections for Transfinit algorithm in the wing tip case.

The main limitation of *gmsh* is the fact it is not a multi-block mesher, so the whole meshing process could require more effort. It provides relatively robust two-dimensional hex-meshing algorithm allowing to easily mesh boundary layers through extrusion but lacks fully three-dimensional recombination method. That is why we did not use the automatic meshing tools splitting instead the computational domain into a set of subsections and applying Transfinit (gmshModelMeshSetTransfiniteCurve) algorithm. This algorithm allows to build volume and surface meshes based on defined split of edges and provides a user full control over the generated mesh but is very labour intensive. However, combined with native Nek5000 mesh smoothing algorithm it allows to produce hight quality meshes optimal for SEM solver. One should mention here Coherence (gmshModelOccRemoveAllDuplicates) operation, which ensures consistency of the mesh removing multiple overlapping objects with the unique one. This operation is crucial as higher-level objects e.g., surfaces can be created as separate objects not sharing edges. For complex meshes Coherence can be time consuming operation as well.

Apart from the global Coherence (gmshModelOccRemoveAllDuplicates) operator there is a local counterpart based on BooleanFragmenst (gmshModelOccFragment) allowing to limit the operation to a specified set of objects and save time. Unfortunately, we found the result of the local operation to be dependent on the type of object it is applied to.

BooleanFragmens are effective with respect of surfaces generated with ThruSection (gmshModelOccAddThruSections), but the surfaces generated with e.g., gmshModelOccAddBSplineSurface would rather require the global operator.

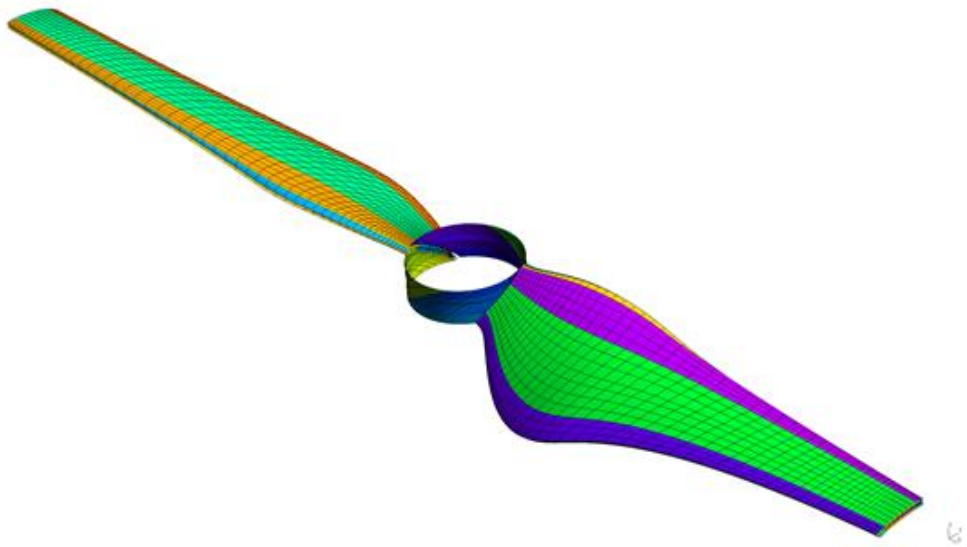

Figure 4: High order mesh produced with gmsh for a relatively complex drone rotor.

The other important aspect is proper representation of the geometry of the complex external surfaces adequate to SEM solver. We found the surface correction step performed within Nek5000 to be crucial. It is even more important for AMR solver as the external surface representation should be correlated with variable resolution. That is why both gmsh and Nek5000 require consistent surface parametrisation. In gmsh we found useful OpenCASCADE [2] kernel and BSplineSurface surface representation. ThruSection algorithm produces surfaces with equal quality, but Coherence operation becomes more challenging. In EXCELLERAT we used both methods successfully generating high quality coarse meshes for AMR Nek5000 solver.

3 Simulation

As the central step for all of the use-cases considered during the initial phase of EXCELLERAT was the simulation step, i.e. the execution of the simulation code on large scale HPC resources, with the main targets being efficiency and scalability towards exascale readiness, most of the work of the EXCELLERAT team was focused on the simulation step.

As the level of parallelism is considered to grow about one to two orders of magnitude, depending on the system architecture, the focus when going from petascale to exascale application scale has to be on the effective decomposition of the discretized simulation domain into suitable parts for the given system architecture.

Since the data size used to discretize the simulation domain significantly grows one will directly find that it is no longer efficient and at some point not even possible to generate the needed meshes offline and transfer the data from the user site to the HPC centre.

The strategy to circumvent the problem of high-resolution mesh generation is, generate lower resolved baseline meshes and then move the generation of the final, highly resolved computational mesh by so called mesh refinement “in core” i.e. to the HPC system. Another step further which can greatly increase the efficiency of the complete simulation step is to use of adaptive mesh refinement techniques in which only the regions of the computational domain are refined in which the targeted physical phenomena show high spatial variations. The diverse strategies to implement AMR into simulation codes that were used in EXCELLERAT are summarized in section 3.4. In addition, overviews of the knowledge gained as best practices in the areas upstream of adaptive mesh refinement, discretisation methods and domain decomposition, are presented in sections 3.1 and 3.2. If implicit discretisation methods are used, then techniques for solving linear systems of equations are another important area which must be considered when scaling problems efficiently towards exascale. The insights gained in this area are summarised in section 3.3. A trade-off that of course comes along with the implementation of AMR methods is the unbalanced load distribution between the parallel processes that arises from decomposition of the initial mesh and local refinements which is why load balancing strategies were investigated in detail and the respective findings that are considered as best practices are presented in section 3.5. As all the aforementioned methods improve significantly the efficiency and scaling of the codes, they as well introduce significant changes into the numerics of the codes. Due to that validation and regression gets an even more important role as it has already. The corresponding best practices that were found by the EXCELLERAT consortium are presented in section 3.6. As one can state that the movement of a problem towards large scale HPC is nowadays first of all the conversion of a compute or memory bound problem into an I/O bound problem, the best practices in that are presented in section 3.7.

3.1 Discretization Methods

Discretization methods rely on the introduction of some kind of grid. This grid can be either regular (uniform) or irregular (non-uniform). Often regular grids are used to discretize domains which are simple from the geometric point of view whilst irregular grids are used for more complex cases. This complexity often arises from the existence of irregular surfaces in the domain. For example, the boundary of the domain may complex or there may be a complex solid body immersed in the fluid (such as a car or an aircraft).

The model associated with a regular grid may have all of its degrees of freedom collocated or they may be distributed. A MAC (or staggered) grid, for example, has scalar quantities located at the centres of cells whilst the components of vector quantities are located on the faces of the cells. This may be done because of the conservative properties of such a discretization. TPLS is an example of simulation software that uses a MAC grid. Uniform grids are usually associated with the use of finite difference methods.
Non-uniform grids are usually associated with finite element methods. They allow the description of complex surfaces through meshing (see subsection 2.2).

3.2 Domain Decomposition

Domain decomposition is often important when using HPC systems because of the need to spread computation over nodes. This needs to be done in a way that balances the load on each node.

For simple domains such as cuboids that are covered by a uniform mesh it is relatively easy to subdivide the domain into chunks of roughly equal size that require similar amounts of time to process, Of course the decomposition of the domain leads to the need to exchange data at the shared boundaries of the subdomains. In developing TPLS it has been found useful to make use of PETSc (https://petsc.org/release/) to manage the decomposition of the domain and the exchange of data. Once the decomposition has been described using PETSc's data structures the exchange of data may be implemented using simple function calls that hide the details of the underlying MPI calls.

The non-uniform meshes produced by finite element methods require more effort to balance the load. Graph partitioning packages such as ParMETIS and PT-Scotch may be used to implement the decomposition. PETSc supports domain decomposition via external packages such as ParMETIS and PT-Scotch.

Larger supercomputers allow the simulation of more complex phenomena with increased accuracy. Eventually this requires finer and thus also larger geometric discretizations, referred as meshes. In this context, and extrapolating to the exascale paradigm, the pre-processing stage of a simulation, which sets up the parallel execution given an initial mesh, becomes a critical part of the workflow.

In EXCELLERAT, BSC has devoted a considerable effort on implementing a fully parallel workflow that has been used on simulations engaging O(10^5) parallel processes. As shown in Figure 5, the pre-processing stage is mainly based on interleaving data redistribution phases with the mesh partitioning. The most critical bottleneck of the pre-processing stage is the mesh

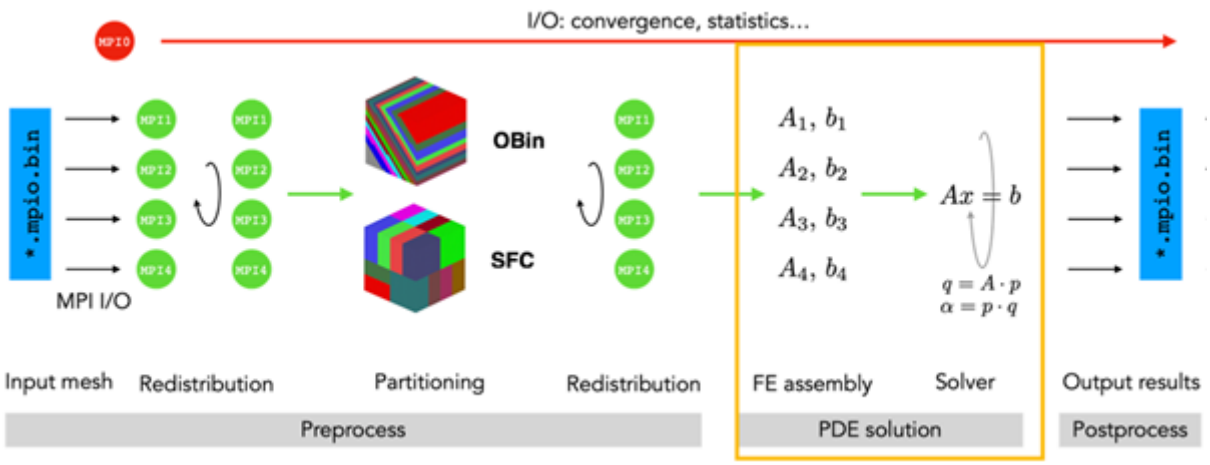

Figure 5: Main phases of a CFD simulation using Alya.

partition. An in-house SFC based approach has been implemented in Alya to overcome limitations of graph-based approaches.

Mesh partitioning is traditionally based on graph partitioning, which is a well-studied NP-complete problem generally addressed by means of multilevel heuristics composed of three phases: coarsening, partitioning, and un-coarsening. Different variants of them have been implemented in publicly available libraries such as Metis/ParMETIS [3], Scotch/PT-Scotch [4] and Zoltan [5]. All these libraries enable parallel partitioning, but with a limited parallel performance and a decreased quality of the parallel partition. Both aspects make graph-based partitioning a potential bottleneck in the simulation workflow. However, considering the evolution of the computing HPC systems, any potential bottleneck becomes an effective bottleneck if not addressed in time. Motivated by these circumstances, in Alya we have implemented a fully parallel geometric partitioning alternative.

Geometric partitioning techniques obviate the topological interaction between mesh elements and perform its partition according to their spatial distribution. If we consider as unitary element the mesh cell, then its mass centre can be used to determine its spatial location. A Space Filling Curve (SFC) is a continuous function used to map a multi-dimensional space into a one-dimensional space with good locality properties, i.e. it tries to preserve the proximity of elements in both spaces. The idea of geometric partitioning using SFC is to map the mesh elements into a 1D space and then easily divide the resulting segment into equally weighted

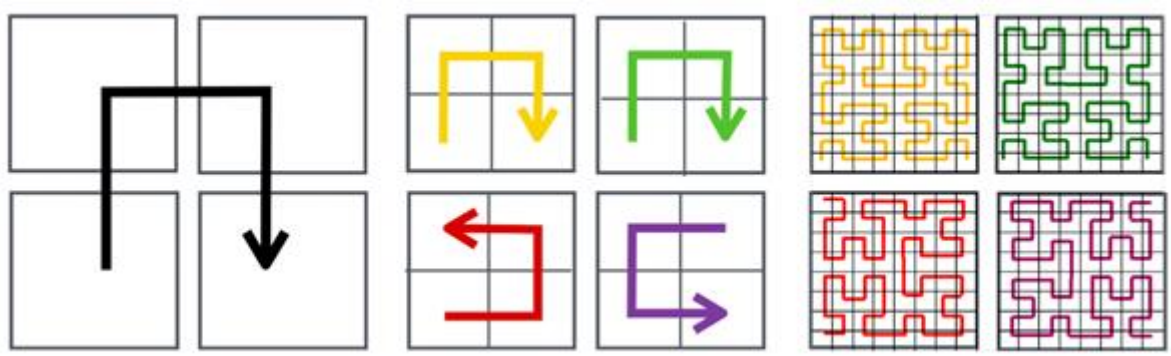

Figure 6: Hilbert SFC parallel generation. (a) Initial coarse grid definition, (b) orientation of each parallel process to continue the recursion, (c) recursive generation of SFC in each parallel process.

sub- segments. A significant advantage of the SFC partitioning is that it can be computed very fast and it is easy to parallelize, especially when compared to graph partitioning methods. However, while the load balance of the resulting partitions can be guaranteed, the data transfer between the resulting subdomains, measured in terms of edge-cuts in the graph partitioning approach, cannot be explicitly measured and thus neither be minimized.

In the course of EXCELLERAT BSC has published two papers [6, 7] describing and evaluating the performance of a novel parallel SFC based partitioner. In the algorithm we avoided any computing or memory bottleneck that could limit the scalability, imposing that the solution achieved is independent (discounting rounding-off errors) of the number of parallel processes used to compute it. An illustration of the process is shown in Figure 6, the SFC is built in parallel allowing to reach the level of granularity required by each problem.

3.3 Solutions Methods

3.3.1 Linear Systems Solvers

The problems (including the use cases of EXCELLERAT) that are tackled on HPC systems are large and unsuitable for solution by direct solvers because of the amount of memory that such a solver would require to function. Consequently, iterative solvers are used. Frequently the iterative solvers used are Krylov solvers. In order to perform well Krylov solvers must be used in conjunction with suitable preconditioners.

The type of Krylov solver that should be used depends on the nature of the problem being solved. For instance, there are some solvers that are suitable only for the solution of symmetric, positive definite systems. For some problems it may be necessary to experiment with different solver plus preconditioner combinations in order to achieve satisfactory performance.

PETSc, the Portable, Extensible Toolkit for Scientific Computation (https://petsc.org/release/) provides implementations of many Krylov solvers and preconditioners that may be used to ease the burden of implementing a solver for a specific problem. Solvers and preconditioners may be chosen at run time which makes experimentation to determine the best combination much easier.

TPLS makes extensive use of PETSc for many things including implementation of solvers. Owing to the fact that the range of problems that can be tackled using TPLS is continually increasing through the addition of models of new physical processes, and the fact that the majority of the people developing TPLS are subject matter experts rather than experts in simulation, it has been found profitable to prove the concept of a new development by first implementing a simple but inefficient solver such as weighted Jacobi before progressing to more sophisticated methods using PETSc.

Use of a toolkit such as PETSc represents a move away from the continual reinvention of the wheel.

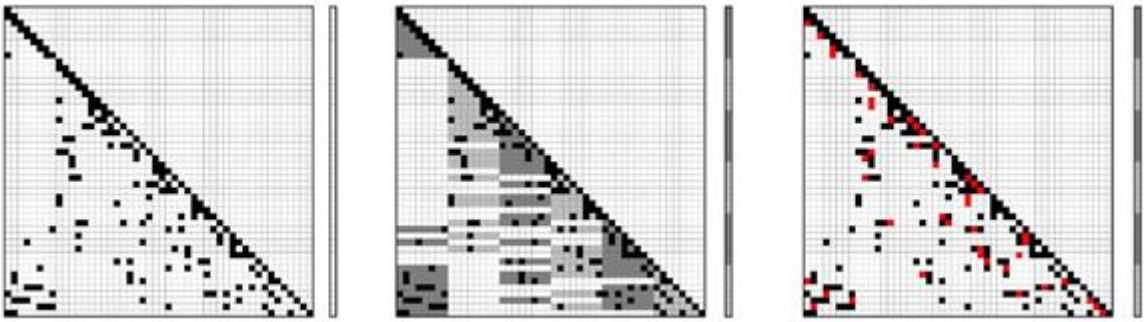

Figure 7: Graphical representation of the pattern extension strategy. Left: Initial lower triangular pattern of a given matrix, A (black squares) plus the multiplying vector x. Centre: Cache-friendly pattern extension. Right: Filtered pattern.

Cache-aware Sparsity Patterns for the Factorized Sparse Approximate Inverse Preconditioner (BSC)

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 [8, 9] are a prominent option with the advantage that, relying on the SpMV kernel for its application, they are easily parallelizable and portable to any computational architecture.

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 increase also 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.

An illustration of this idea is shown in Figure 7. Given a sparsity pattern based on numerical criteria (left), all the entries of the matrix that do not generate new cache misses on the multiplying vector are considered to extend the pattern (centre), finally entries with small value are filtered out (left). The additional entries of the extended pattern do not generate new cache misses. A detailed description of this new algorithm proposed by BSC and referrer as FSAIE was presented in the 30th International Symposium on High-Performance Parallel and Distributed Computing [10].

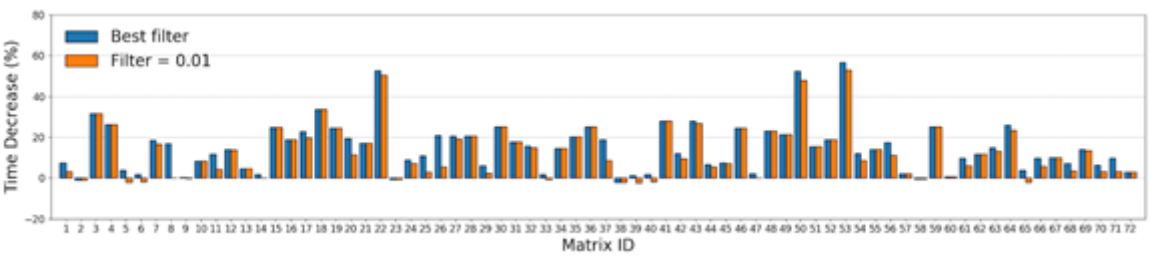

Figure 8: Time decrease of the FSAIE (full) vs FSAI using the best $filter$ value per matrix (blue columns) and $filter$=0.01 value (orange columns) on the Skylake architecture.

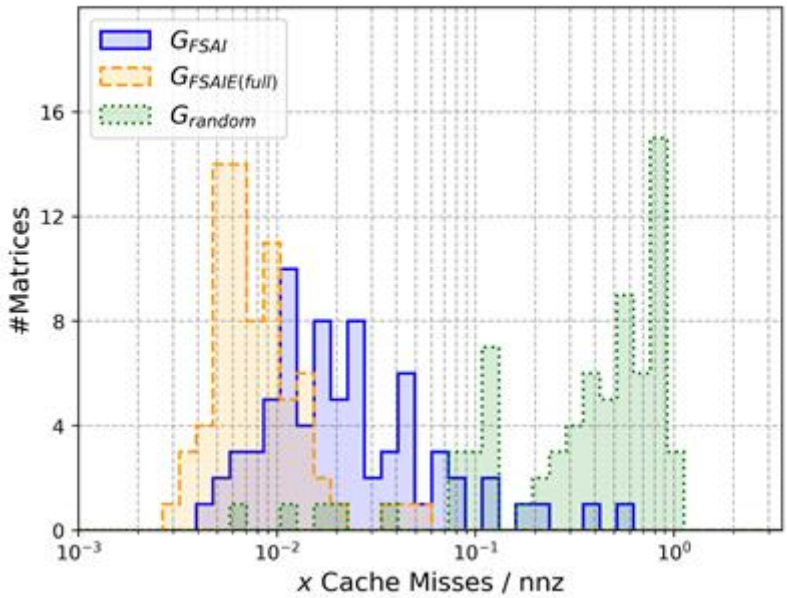

Figure 9: Histogram of L1 data cache misses on the application of the inverse preconditioner for 72 matrices considered in the SuiteSparse Matrix Collection.

An extensive evaluation campaign considering 72 matrices of the SuiteSparse Matrix Collection was performed. It demonstrates on the Skylake architecture average improvements of 15.02% in terms time to solution, and time reductions of more than 50% for some matrices. We asserted that the gains come from a better utilization of L1 cache level (Figure 9).

The FSAIE algorithm was also tested on ARM CPUs. The large 256 Bytes cache lines of A64FX produced the best results, namely, average improvements of 22.85% in terms time to solution, and time reductions of more than 75% for some matrices, see Figure 6.

In summary, we have observed that cache-aware optimizations produce substantial benefits across large sets of matrices and on different architectures.

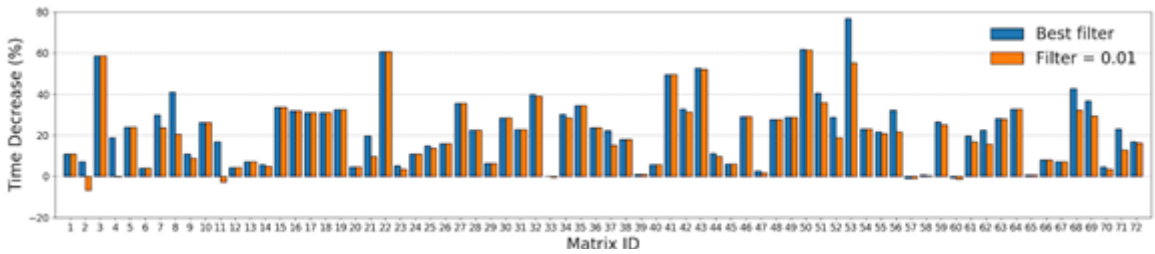

Figure 10: Time decrease of the FSAIE (full) vs FSAI for the best *filter* value (blue columns) and for the 0.01 *filter* value (orange columns) on the A64FX architecture.

3.4 Dynamic Mesh Adaptation

Adaptive Mesh Refinement (AMR) is a method to ensure a computational mesh has more elements in regions of interest (more turbulent) and has fewer elements elsewhere (more laminar). This subsection describes the lessons learned regarding their own AMR libraries by four of the EXCELLERAT partners, namely RWTH, BSC, CERFACS and KTH.

RWTH have developed AMR within their m-AIA application. They employ a regular octree mesh, and AMR is triggered based on phenomenon-based sensors embedded within the target simulations. Moreover, these sensors help determine if mesh cells should be refined or coarsened.

BSC have parallelized their AMR workflow within their Alya application. Alya (gitlab.bsc.es/alya/) employs unstructured meshes, using the gmsh mesher. The mesh is partitioned using Metis and an in-house space-filling curve (SFC) partitioner. The solver uses first- and second-order finite elements. Alya can be used to solve 3D incompressible Navier-Stokes flow, low Mach flow, multiphase flow, and combustion. Numerical schemes include low-dissipation numerical schemes for momentum and scalar transport, entropy-stable algorithm for multiphase flow based on a conservative level set, explicit time schemes, Runge-Kutta 3rd and 4th order, and Krylov subspace solution methods for pressure solver

CERFACS have created a new AMR process for their AVBP application, namely TREEADAPT. AVBP (www.cerfacs.fr/avbp7x) employs unstructured meshes, using the gmsh icemcfd, centaur soft, and cfd-geom meshers. The mesh is partitioned using ParMETIS, Ptscotch, Metis, and an in-house implementation of RIB and RCB. The solver uses a combination of finite difference, finite elements and spectral elements in second- and third-order. The code can be used to solve 3D compressible Navier-Stokes flow, with LES for reactive flows. Numerical schemes include 2nd (Lax Wendroff) and 3rd order (Taylor Galerkin) explicit schemes. TREEADAPT was built on top of the hierarchical domain decomposition library TREEPART and is able to use the computing system topology for optimal parallel performance. Mesh adaptation itself is handled by the opensource package MMG who is capable of adapting triangular and tetrahedral elements.

KTH describes AMR spectral method application, Nek5000. The AMR method does not employ re-meshing but increases the number of degrees of freedom by increasing the element count. Nek5000 (nek5000.mcs.anl.gov) (gitlab.bsc.es/alya/) employs unstructured meshes, using the following meshers: native simple tools; gmsh (supported converter gmsh2nek); additional supported converters from Exodus format (HEX20 in 3D and QUAD8 in 2D) and CGNS library (HEXA8, HEXA20 and HEXA27). The mesh is partitioned using ParMETIS, and an in-house parRSB partitioner. The solver uses spectral elements of up to 15th-order. Nek5000 can be used to solve 3D incompressible Navier-Stokes flow. Numerical schemes include time-stepping BDF/EXT; a pressure correction scheme witch staggered pressure points (PN-PN-2), and a 2-level additive Schwarz pre-conditioner for pressure equation.

3.4.1 RWTH: Adaptive Mesh Refinement for Hierarchical Cartesian Grids

Based on the EXCELLERAT work by RWTH on re-meshing of hierarchical Cartesian grids the following approach on meshing and AMR using phenomenon-based sensors have proven to be computational efficient and to yield good results. The sensors are placed on grid points and can be, e.g., measures of vorticity or entropy. The experience is based on the spatial discretization approach using hierarchically-refined Cartesian meshes implemented in RWTH's own multi-physics m-AIA simulation framework.

In the hierarchical Cartesian mesh, the cells are organized in a cell-tree structure, which is based on parent-child relations. The grid generation starts with a single cubic cell, which encloses the entire computational domain of all coupled systems and is the root of the cell-tree data structure. This zero-level cell is recursively refined by being subdivided into eight cube-shaped child cells. Each cell can be separately refined or coarsened, regardless of the refinement level of surrounding cells. In the process of cell refinement, a cell is split into child cells and becomes a parent of these. Unrefined cells are referred to as leaf cells. Parent–child relationships exist between cells at different refinement levels and neighbour relationships exist between cells at the same level. The cell-tree data structure is completely stored from the leaf cells on the highest level of refinement all the way up to the cells of the isotropic start grid, i.e., parent cells are not deleted from the data structure. The data structure takes full advantage of AMR since coarsening operation can be performed without (re)creating the coarse cells.

Experience has shown that an AMR based on a phenomenon-based approach produces among the best results. That is, sensors are employed to detect and localize physical flow phenomena, and hereafter, a grid refinement is initiated where appropriate. The utilized sensors for mesh refinement can, e.g., be defined on the magnitude of the vorticity vector, consequently detecting shear layers or on a measure for the entropy detecting phenomena that generate entropy gradients as, e.g., shock waves. The sensors are weighted by the cell length size to take into account the present cell refinement. The sensors are computed at given time steps or on specified conditions for the cells, and refinement or coarsening of a cell is controlled by minimum/maximum threshold values which, in turn, are based on the statistical distribution of the sensor in the flow field.

The adaptation process can be carried out after a time step is completed, and there are several criteria can be used to determine whether the adaptation procedure should be started or not. For steady-state problems a residual-based criterion has proven to be meaningful, i.e., adaptation is initiated every time a specified residual convergence limit is reached up to a maximum number of adaptation steps. For unsteady problems, e.g., for a von Kármán vortex street, a time-step-based criterion is reasonable, such that the grid is adapted at some regular time step interval (see Figure 11).

In steady state problems, studies have shown that it is advantageous to impose a constraint that, at each adaptation step, the number of deleted cells must not exceed the number of newly created cells such that the overall number of cells is not decreasing. In doing so, oscillations of the number of cells in the limit of grid convergence can be avoided. In unsteady cases, it is

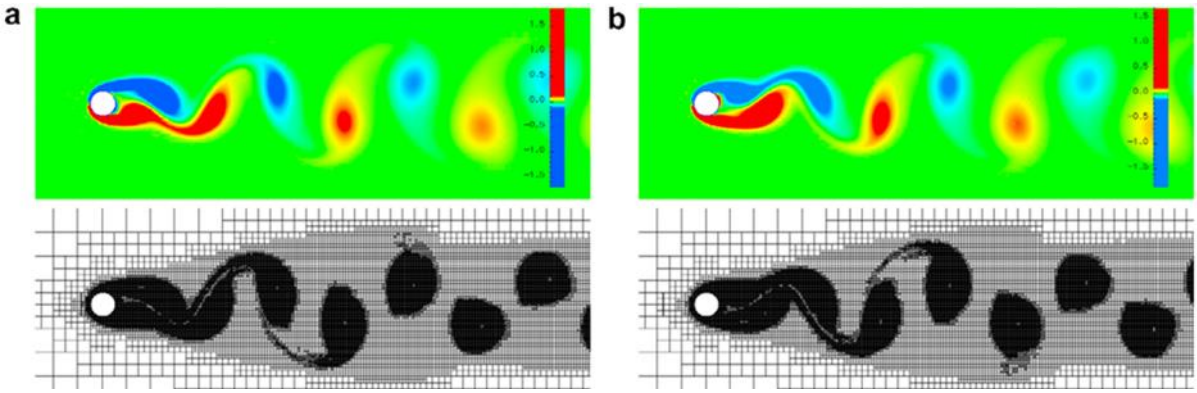

Figure 11: Vorticity distribution and corresponding solution-adapted grids for the simulation of the unsteady flow around a circular cylinder.

desirable to keep the number of grid cells constant and, at the same time, to enable a dynamic change of the grid according to the solution. The adaptation parameters, i.e., the minimum and maximum threshold values, that determine how many of the cells are refined or coarsened must be selected correctly to enable the grid changes required to dynamically follow the solution (see Figure 11).

3.4.2 BSC: Implementing AMR into Alya

Within EXCELLERAT, BSC implemented a parallel AMR within the simulation code Alya.

The steps to migrate the simulation from one unstructured mesh into a new one are illustrated in blue in Figure 12. First the error obtained due to the former mesh is estimated in order to measure the adaptation requirements, then a new mesh is generated in parallel. Finally, the solution is interpolated from the former mesh to the new own. After this process, the domain decomposition may have become unbalanced since the refinement/coarsening is locally determined by the physics evolution. If this is the case, a load balancing step is required, this optional step is illustrated in green in Figure 12.

For error estimation, different strategies have been implemented. Tests revealed that basing the error evaluation on a Laplacian filter was the most robust approach. However, the error estimation is in general a physics dependent aspect, that requires problem-dependant specific treatment. Once the error is evaluated, a sizing formula, provided by *gmsh*, can be used to evaluate a *size field* as an input for gmsh to generate the new mesh. The size field is the target mesh size for each zone within the domain and is based on the given error estimate and the total number of elements of the mesh.

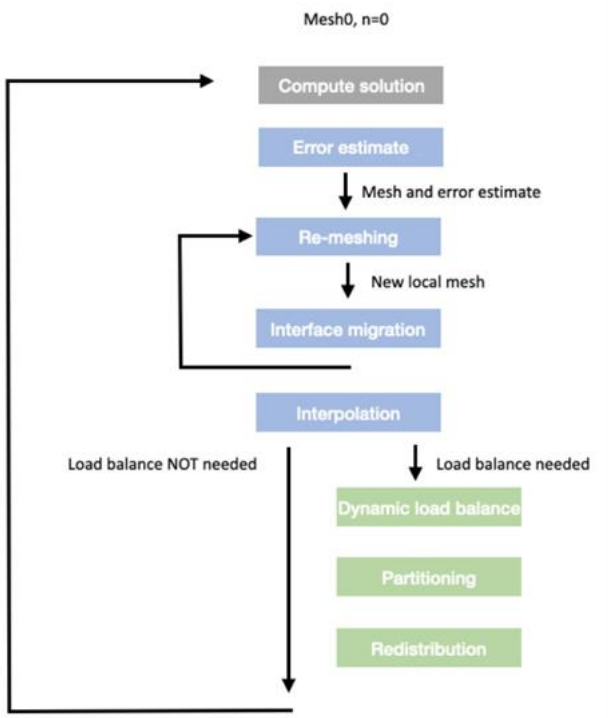

Figure 12: AMR parallel workflow.

Regarding the mesh generation, Alya is linked with the open-source mesh generator gmsh. Each parallel process generates a new mesh within the boundaries of its subdomain's surface. The

main issue with the parallelization of the re-meshing is to deal with the mesh generation at the subdomains' borders. Since a conformal approach has been implemented in Alya, the interface elements between subdomains must match. To ensure this, an *interface freezing* approach has been adopted. In this approach, the interface elements are not changed such that each process can adapt the rest of its subdomain without developing incoherent mesh configurations at the interface. However, this approach requires an iterative process, interleaving displacement of the interface and local remeshing, to ensure that the overall domain is re-meshed according to the adaptation requirements.

Once a new mesh is generated in parallel, a parallel 3D interpolation is performed to migrate the solution from the former to the new mesh. This interpolation is based on point-to-point communications and the most expensive part is the evaluation of the interpolation coefficients that requires searching the element of the former mesh containing each node of the new mesh.

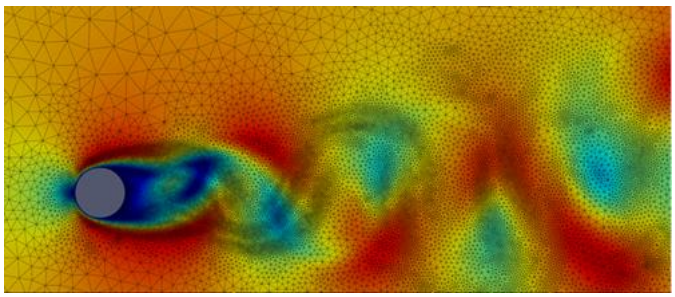

Figure 13: Mesh adaptation for the flow around cylinder.

Regarding the Load Balancing, as described above, this is an optional stage that will depend on the load imbalance of the resulting mesh. A threshold is used to avoid the overhead of repartitioning the mesh when the imbalance is low. In Alya the partitioning is based on a parallel Space Filling Curve (SFC) method which can be very fast. Graph based approaches tend to provide better solutions but at a much higher cost.

Figure 13 presents an illustrative snapshot of the mesh adaptation for the simulation of the flow around a cylinder at *Re*=120. It can be observed that the mesh is concentrated around the vortex structures of the velocity field.

3.4.3 CERFACS: AMR Strategy for AVBP

Within EXCELLERAT CERFACS developed an AMR strategy for its solver AVBP. The AVBP code relies on unstructured grids for complex geometry and uses domain decomposition for parallelization. At the beginning of EXCELLERAT no parallel adaptation framework existed. However, extensive experience existed at CERFACS using the MMG library, therefore we focused on parallelizing MMG instances for our usage.

Two methods were implemented during EXCELLERAT: first, the coupling with the proprietary library YALES2 [https://www.coria-cfd.fr/index.php/YALES2] and secondly a home-made alternative TREEADAPT based on a the TREEPART hierarchical mesh partitioning library developed by CERFACS for the EPEEC project.

The AMR strategy of both frameworks is displayed in Figure 14, and is as follows:

- Initialisation: An initial partitioning level must be provided to the adaptation framework.

- Adaptation step: each individual domain is adapted using MMG, which freezes the borders that correspond to internal domain overlaps to keep matching elements between partitions. This adaptation is performed using a user defined metric stored at the vertices of the mesh which is physics dependent and translates to the length of the attached edges to a given node.
  Here, YALES2 and TREEADAPT differ. YALES2 adds an internal loop at this point at the individual domain level where the skewness of the adapted mesh is checked and adapted until satisfaction. TREEADAPT does not include this feature as this is not currently required for AVBP's explicit solver.
- Interpolation: the current solution fields are interpolated to the next mesh. YALES2 uses a linear interpolation whereas TREEADAPT uses a least squares algorithm.
- Load balancing: after the adaptation, a load balancing step is performed to distribute the resulting mesh. NB the previously frozen frontiers are weighted to ensure they become internal for the next adaptation step.
- The current mesh is then checked against the targeted metric. If the requirement is not met then the adaptation, interpolation and load balancing steps are repeated until convergence is achieved, and the adaptation framework can be exited.

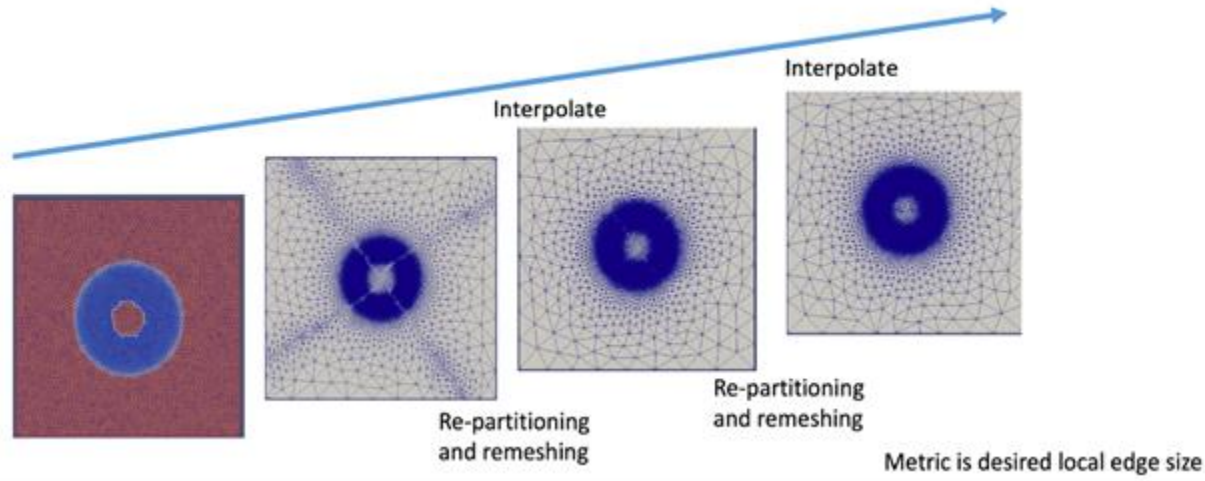

Figure 14: Visualisation of the mesh adaptation steps on a 2D case.

3.4.4 KTH: AMR for spectral code Nek5000

Within EXCELLERAT KTH developed an AMR framework for high-order CFD solver Nek5000 based on a spectral element method (SEM), in which a computational domain is decomposed into a set of non-overlapping, loosely coupled subdomains called *elements*. Each element is treated as a spectral subdomain with the approximation space spanned by the Lagrangian interpolants of order *N*.

In our approach to AMR we do not preform re-meshing but exploit existing domain decomposition resulting from the SEM and increase the number of degrees of freedom by increasing the number of elements. This is achieved by a simple octree splitting in which a single three-dimensional element (so-called "parent") is replaced by its 8 "children" elements. A big advantage of this strategy over, e.g., a local variation of a polynomial order, is a smaller constraint on the Courant–Friedrichs–Lewy (CFL) condition and more localised refinement as

the octree refinement can be recursively repeated. On the other hand, this method introduced so-called hanging nodes and, as such, the solver was modified by adding an interpolation operator at nonconforming faces. (In general, SEM requires an interpolation operator and its transpose on nonconforming faces. However, previously, it was found via numerical experimentation that Schwarz based pressure pre-conditioners were more effective when using an inverse operator.) Although this introduces some work imbalance, numerical experiments showed its effect to be negligible, and both conforming and nonconforming Nek5000 solvers share the same parallel properties.

Nonconforming meshes may be generated by AMR. The direct way to generate these is to take a conforming mesh and to shift one element layer with respect to the other, thus one loses one-to-one correspondence of faces and the vertices can be located at any face position. The downside is that we lose the simplicity of conforming meshes, wherein data transfer between element faces is a simple copy. To perform data transfer in general nonconforming meshes we require a special operator connecting elements, namely a *mortar* element. In our approach, we simplify a problem by retaining the initial conforming mesh and providing restriction to the refinement method, thereby replacing a “copy” with “interpolation” and thus avoid complex data treatment between mortar elements.

We stress that in our approach we use simple interpolation operators instead of mortar elements, that constrains the refinement freedom, but on the other hand this enables more simple and efficient mesh managers to be employed, such as, e.g., the *p4e*st library [11]. Our current experience shows this conforming-space/nonconforming-mesh approach is not a real limitation and allows to achieve a significant reduction in the simulation cost.

The other important design decision taken was removing the mesh management from the CFD solver itself. In this case p4est provides to Nek5000 information required to construct the mesh built of the children elements only, and all the parent-children relations are hidden (parent elements are never constructed). This reduces the number of Nek5000 modifications and allows to achieve an optimal solver set-up for a given mesh; however, this requires a solver restart after each mesh modification which introduces a cost overhead. This cost is dominated by the grid partitioning and the coarse grid solver set-up. Although, for relatively small simulations this additional cost is negligible, it becomes important for the bigger simulation with MPI rank count reaches tens of thousands. That is why our implementation is currently not well suited for tracking flow features, but it works very well for cases where the final steady mesh can be reached, and this includes multiple statistically stable flow cases.

The last aspect is the error indicator/estimator and the resultant refinement strategy to be employed. It is the most critical point for reducing computational error and minimizing cost. To measure the computational error, we used two tools: a spectral error indicator and a goal-oriented adjoint-based error estimator. The adjoint error estimator comes with a significant simulation cost overhead and problems with simulations of turbulent flows. (At first glance, adjoint solvers can be describe as integrating backward in time. To get solution sensitivity one has to integrate forward, using a direct solver, and then backward, using an adjoint solver. For laminar flows, the adjoint step is usually simple, but turbulent flows are inherently chaotic and adjoint steps can lead to numerical instability without workarounds.) As such, the spectral error indicator was used in most of our production runs, as it was found to provide sufficient mesh

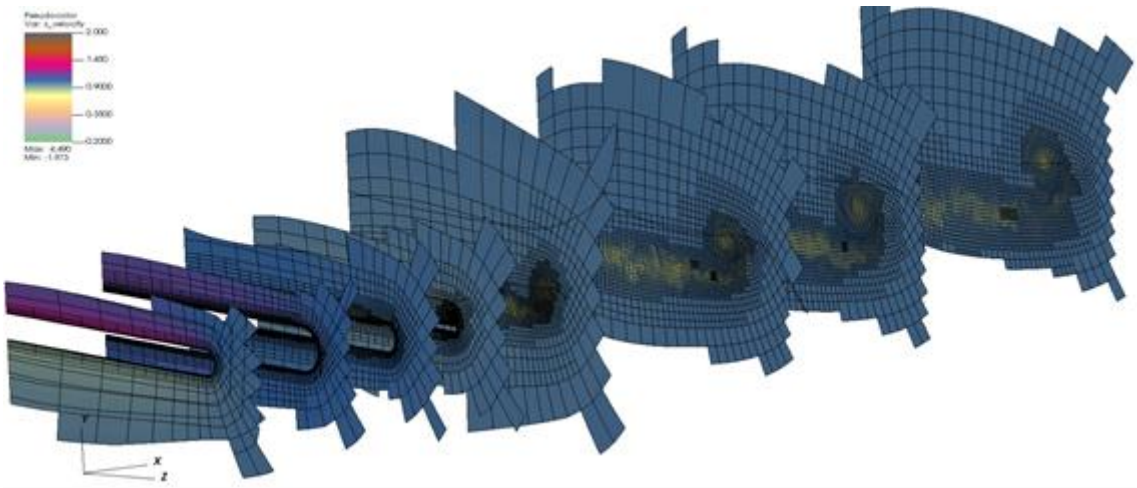

Figure 15: Mesh structure for the AMR simulation of the flow over NACA0012 wing profile with rounded wing tip. Element borders are marked with black lines.

quality. To target the chaotic nature of turbulent flows we currently work on an error estimator based on the transfer entropy measuring the causality between the quantities of interest, such as drag or lift, and flow variables, such as velocity and pressure.

Our implementation of AMR framework in Nek5000 has been very successful, allowing us to perform previously unaffordable high-fidelity simulations of, e.g., the wing tip or the simplified rotor (see Figure 15).

3.5 Load Balancing

The need for load balancing arises from domain decomposition as discussed in section 3.6 Domain Decomposition.

WP3-BSC Intra-node and system level load balancing strategies implemented in Alya

The application of a dynamic load balance strategy for the integration of stiff chemical source terms in combustion simulations with detailed chemistry was addressed. Chemical reactions occur in thin layers, which are usually characterized by highly non-linear and stiff chemical reaction rates that are very costly to evaluate. In fact, this problem is of relevance when more realistic fuels or surrogates are to be considered. In Alya, the source terms are integrated using an implicit first-order backward Euler scheme based on the library CVODE [12]. The computational costs of the chemical integration take a large share of the overall timestep for complex fuel. Therefore, strategies for increasing the performance are of paramount relevance. The high imbalance is inherent to the problem being solved. The chemical reaction is only solved at the elements containing the flame and for a certain temperature range. Therefore, the imbalance cannot be addressed by repartitioning the domain, because the problem is unsteady, and the flame undertakes some dynamics that cannot be predicted. In collaboration with POP (https://pop-coe.eu), the DLB library (Dynamic Load Balancing library) [13] has been integrated into Alya to reduce the imbalance and to increase the computational performance in combustion simulations with stiff chemistry.

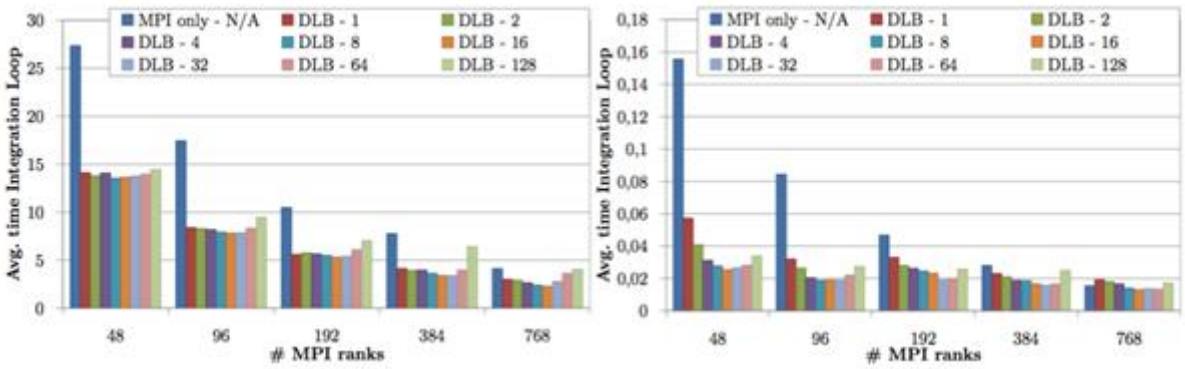

Figure 16: Scalability of the chemical integration loop: detailed chemistry (left) and reduced chemistry (right).

In Figure 16, the duration of the integration loop for the detailed and reduced chemistry cases, where the x-axis represents the different numbers of MPI ranks corresponding to 1, 2, 4, 8, and 16 nodes on MareNostrum IV supercomputer, is plotted. Obviously, the DLB integration improves the pure MPI code in all cases. It is furthermore observable that the impact of the grain size becomes more important when the number of MPI processes is increased. A high grain size value has a negative impact on the performance when DLB is used and the number of MPI ranks is increased. This is due to the fact that the increase of the number of MPI ranks leads to a lower load per rank and packing the load in big chunks does not allow for malleability to balance the load by DLB. The optimum grain size in all the cases is found to be around 32. In the reduced chemistry case using small or large grain sizes has a negative impact on the performance. That is, in the case of large grain sized the same situation as in the detailed chemistry use case appears, i.e., large grain sizes do not offer sufficient flexibility to load balance the computation. As it can be seen in Figure 16, using small grain sizes in the reduced chemistry case leads to a reduced impact of DLB on the performance due to the smaller relative weight of the integration loop in the whole time step.

The speed up obtained in the integration loop using DLB with different grain sizes is compared to results of a pure MPI version running with different numbers of MPI ranks in Figure 17. For clarity, only the largest and smallest grain sizes (i.e., 1 and 128) and the ones that delivered best performance (i.e., 16, 32 and 64) are shown. From the results it becomes obvious that with DLB using a grain size of 16 the execution can be accelerated by a factor of 2x in all the cases for the detailed chemistry case, except when using 16 nodes (768 MPI ranks), where a speed up of 1.8x is achieved. It is observed that the more nodes are used the less speed up DLB is able to

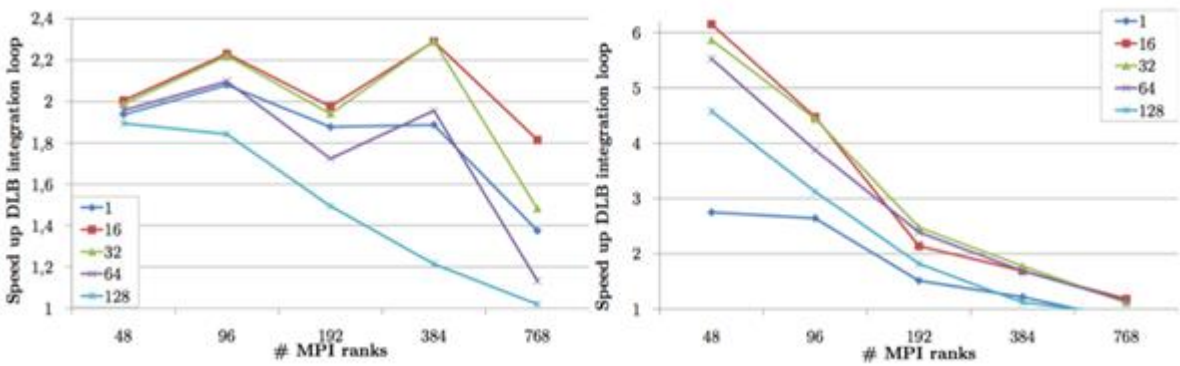

Figure 17: Speed up of DLB compared to the pure MPI execution for the integration loop: detailed chemistry (left) and reduced chemistry (right).

obtain. This is due to DLB not featuring load balancing across nodes and also because the amount of computation per MPI rank is reduced. This leads to less granularity hindering optimal load balancing. For the reduced chemistry case, the speed up using 48 and 96 MPI ranks (1 and 2 nodes) is more impressive than in the detailed chemistry case. Here, the speed up reaches factors 6x and 4.5x. This is because the load balance in this case is fairly low, leaving a lot of space for improvement by applying DLB. When the problem is partitioned among more MPI ranks, the load is more distributed, leaving less load imbalance to address by DLB.

3.6 Validation

Scientific and engineering simulations need to be validated in order to demonstrate their relevance.

Comparison to theoretical results should be made when such results are available. Of course, complete theoretical results will not be available (as the existence of such results would make the use of simulation redundant) but partial results such as linear theory for small disturbances may be available.

Comparison to experimental results should be made when such results are available. Use cases may be chosen to correspond to experiments that have been made or are being carried out in parallel with the simulations.

Visualisations may be used to check that the results produced by simulations are qualitatively correct.

All of these methods have been used in the development of TPLS. For the case of TPLS ParaView (https://www.paraview.org) has been found to be a useful visualisation tool.

3.7 Parallel I/O

Libraries for parallel input and output, such as HDF5 and NetCDF, exist and should be used. They are built on top of MPI I/O and provide higher level interfaces. As well as providing performance benefits over serial I/O they make some simulations possible that would otherwise be impossible. This is because serial output is implemented by gathering all of the data in a single MPI process and then writing it from there. The amount of memory available on a node may not be large enough to allow this. Similar considerations apply to serial input.

Checkpointing should use the same I/O mechanism as the rest of the I/O in the program. A single checkpoint only is required to restart a simulation. Checkpoints consume disc space and it is usually desirable to delete then as soon as possible. However, to allow for the possibility of a program terminating whilst writing a checkpoint, the previous checkpoint should not be deleted until the new checkpoint has been completed.

4 Post-Processing

Once the simulation step i.e. the simulation code is executed or produces results, the output data have to be post-processed in order to be interpreted by either a domain expert or in case of e.g. an automatic optimization procedure an algorithm. In both cases, especially when data are stored on file systems or have to be transferred to another system or the user's site, the first thing to do is to apply data reduction and compression algorithms. The findings and best practices with respect to this are presented in section 4.1. When targeting exascale applications, looking at the produced raw result data of the simulation fields is no longer the way to go even though visualization might be possible. Instead, a key methodology that has to be employed in order to evaluate and analyse the produced large scale result data and to draw meaningful insights is HPDA. An overview about the best practices in this area can be found in section 4.2 while the visualization methods that were found to be best applicable in the case of exascale engineering applications and especially in combination with HPDA are presented in 4.4.

4.1 Data Reduction and Compression Algorithms

Over the years, a steady increase in computing power has enabled scientists and engineers develop increasingly complex applications for machine learning and scientific computing. While these applications promise to solve some of the most difficult problems we face today, their data hunger also reveals an ever-increasing I/O bottleneck. Therefore, as the complexity of large-scale applications and HPC systems increases, it becomes essential to provide users and developers with tools that make it easier for them to get the last ounce of I/O performance out of modern computing clusters.

To address the problem at its inception, it might be useful to think about reducing the amount of data that needs to be transferred. One strategy to mitigate this I/O bottleneck would be to reduce the spatial and temporal resolution of the simulation files. Yet, this trivial method of data reduction is acceptable and applicable only in very few cases. Accordingly, the best way forward is to use the statistical redundancies of our numerical datasets to reduce the overall size of the files passed to the I/O stack.

Since effective data storage is a pervasive problem in IT, much work has already been put into refining it. So-called lossless dictionary encoders represent a simple approach to compressing generic datasets. For example, novel entropy encoders based on asymmetric number systems [14] can enable real-time data compression without affecting the compression performance itself (e.g., Zstandard [15]).

However, lossless compression techniques are limited in their performance and can only achieve a size reduction of 10-30% on average. Moreover, these encoders only affect the statistical redundancies of the underlying bitstream and are not able to exploit spatially correlated information. Efforts have already been made to apply lossy compression algorithms from the entertainment industry to floating-point arrays that take these spatial redundancies into account. For example, the BigWhoop [16] compression library was developed as part of the EU projects ExaFLOW and EXCELLERAT to leverage the power of the JPEG 2000 standard for numerical datasets. BigWhoop has already shown, with DNS data from a turbulent boundary layer, that compression ratios of 1:20 are possible without inducing a significant error - of the order of 1% - in the dataset [17].

Contemporary compression strategies for numerical datasets, such as the ZFP algorithm developed by Peter Lindstrom [18] or the JPEG adaptation by Loddoch & Schmalzl [19], take a similar approach to lossy data reduction. However, the reliance on a block-coding

transformation leads to severe compression artifacts at higher compression rates, resulting in an unacceptable loss of information for sensitive simulations.

4.2 Data Analytics

The computed statistics of turbulent flow simulations are generally uncertain due to the finite averaging time. The techniques developed for estimating such uncertainties, [20, 21], act in an offline mode, meaning that they require to have access to all available samples of a time-series at once. This can lead to I/O deficiencies and requirement of having a large storage for large-scale simulation for turbulent flows. Through a collaboration between KTH and Fraunhofer SCAI within EXCELLERAT, we have designed, implemented and tested an in-situ framework to estimate uncertainties in turbulence statistics due to finite time-averaging [22]. For this, we have proposed a low-storage updating formula for autocorrelation function for turbulence time-series. The framework is tested using Nek5000 as the flow solver that is linked to UQit [23] through a VTK-Catalyst interface. The resulting uncertainty estimates are the same as those in the offline mode and the computational overhead added by the in-situ algorithm is negligible. These promising results are encouraging for further development of the in-situ framework for other UQ and data-driven analyses relevant to large-scale flow simulations.

4.3 Data Transfer (from HPC Site)

For the data transfer back from the HPC site to the user's site the same mechanisms for data handling and security should be used as for the transfer to the HPC site even though the facilitation of data compression and reduction methods in combination to the techniques described in section 2.1play a much more prominent role since data size increases significantly.

4.4 Visualization Methods

Visualizing the data that results from (pre-)exascale simulations introduces new challenges, most importantly the huge amount of data produced. Loading this data into a visualization program will require more memory than workstations can offer and even exceeds the limits of dedicated visualization clusters. Using secondary clusters also introduced problems of moving the data. A solution to these issues is remote visualization on the computation cluster.

But sometimes the trouble even starts before the data is written to disk. Data I/O of massively parallel simulations easily reaches the limits of today's filesystems. Writing results do disk is therefore often the bottleneck of exascale simulations. This can be solved through smart reduction of output data during simulation run-time, which is achieved through in situ post processing.

4.4.1 Remote Visualization

This approach is meant to utilize the compute cluster of the simulation also to do the visualization. The advantages are that the data does not have to be moved and the maximum amount of memory is available. The challenge to overcome is that these systems usually do not have sophisticated graphical capabilities and output devices and are in general not locally reachable. Therefore, remote visualization is introduced as a solution. The spectrum of remote visualization methods spreads between these two basic approaches:

- Object based: the data objects required for post processing and visualization are generated and reduced on the compute cluster and then sent to a visualization cluster or workstation for rendering and analysis.

- Image based: the simulation data is visualized and rendered on the compute cluster and the image data is sent to a visualization cluster or workstation for output.

The advantage of the object-based approach is that the image is rendered locally and therefore changes of perspective can be applied without latency. However, scalability is limited because the amount of data that has to be transmitted is proportional to the raw simulation data and can therefore quickly exceed network and local memory capacities. Compression of geometry and mapped data (e.g. with Big Whoop for structured grids) can reduce but not necessarily solve that problem.

In contrast the image-based approach's data transmission consists of image data of a size that is decoupled from the original data size and only determined by the requested resolution. Especially for exascale simulations this is a huge benefit. The downside is that rendering must be performed on the compute cluster which might not be equipped with GPUs. Also, latency is introduced because user input has to be propagated to the remote cluster to receive updated images.

Our approach seamlessly integrates both, object and image-based methods. The first is realized in the sense that the user can cut off the post processing pipeline at any point and connect it to one on a local machine. The latter is implemented as parallel, CPU based remote rendering. Latency hiding through various re-projection methods allows fluid use even in 3D virtual environments while required bandwidth is reduced by image compression algorithms. Furthermore, remotely rendered components can be put into context with locally rendered geometries to restrict latency to the image parts that require remote rendering (because of their data size or location).
Fore a detailed description see: https://elib.dlr.de/93169/1/cresta_whitepaper2_OCT14_03.pdf, section 4.2.

4.4.2 In-Situ Visualization Methods

To analyze simulation data during runtime, simulations must expose their data through a specified interface to the post-processing back-end. While there are in situ interfaces proprietary to HPC-visualization tools like ParaView, VisIt and Vistle, there is also the SENSEI interface that combines its simulation interface with these and other back-ends to allow maximum flexibility on the analysis side. Implementing an in-situ interface in a simulation requires programming effort and using it and can introduce additional dependencies and can lead to overhead through data conversions and copying. Choosing the right interface(s) should therefore be considered carefully depending on the data format(s) of the simulation code and the users' requirements on the back-end(s). Table 1 gives an overview about the used data formats and the required dependencies.

While the LibSim interface provided by VisIt is quite pure and efficient by only sharing pointers and enum types between the simulation and VisIt this approach also leaves more room for errors in terms of type safety and data ownership.

Interface	***Data formats***	***Additional Dependencies***
ParaView (Catalyst)	VTK based	VTK build by ParaView
VisIt (LibSim)	Raw pointers with support for AoS and SoA	Runtime linking to a single LibSim library
Vistle	Proprietary data format based on shared memory arrays with support for SOA. Additionally features a LibSim interface.	Vistle
SENSEI	VTK based	Dependencies to the enabled back-ends and in older versions VTK as eventually used by the back-end.

Table 1: Comparison of the data formats and dependencies of in situ interfaces.

The Vistle interface on the other hand forces the simulation to use Vislte's API to allocate or copy its data in shared memory (shm). This API requires data in the struct of arrays (SOA) format. This is not only higher development effort but also comes with potential runtime costs in terms of memory consumption and data conversion. The advantages are, that Vistle can run as a separate process and therefore prevent the simulation from crashing in case the back-end does. Also, data ownership is no issue since the shm-arrays are managed by Vistle.

The Catalyst (by ParaView) and SENSEI interfaces provide a middle way between VisIt's and Vistle's approach through using VTK as a bridge between the simulation and the back-end. Since VTK offers great flexibility in terms of accepted data formats, mapping between them and data ownership the implementation effort is comparably low while the runtime overhead stays minimal. The only disadvantage is the additional dependency on VTK and in case of SENSEI a potentially additional conversion from VTK to the data format the specified back-end requires. At least this conversion is handled, transparently to simulation developers, under the hood of SENSEI.

Another aspect of in situ analysis is using the knowledge gained during runtime to steer the simulation. Especially LibSim provides custom commands that can be implemented and set by the simulation and then triggered from VisIt (or Vistle using its LibSim interface). These commands can have a string as argument that is then passed from the back-end to the simulation. Even though in theory this can be abused to also send more complex data like bounding conditions to a simulation, these string-based commands are clearly not meant to. And while these commands always must be implemented by the simulation developers, complex commands also require support from the used back-ends.

SENSEI provides functionality to retrieve mesh-based data from the back-end similar to how the back-end access the simulation data.

5 Simulation Workflow & Result Feedback

As described in the previous sections, data transfer of input and result data from and to the user's site is considered in pre- and post-processing steps. In the simulation workflow and result feedback step the focus is directed towards automatic workflow processing. In the first phase of EXCELLERAT especially methods for uncertainty quantification were considered. The best practices in that area are summarized in the following section.

5.1 Uncertainty Quantification Methods

During EXCELLERAT, KTH has been contributing to the method and software development as well as application of various uncertainty quantification (UQ) techniques to CFD problems, in general, and scale-resolving simulations of turbulent flow, in particular. For the latter, the main constraint to deal with is the high computational cost of the simulations which imposes specific requirements on the UQ techniques. A main contribution of KTH has been developing, UQit, an open-source Python package for UQ in CFD [23]. Currently, there are techniques such as standard and probabilistic polynomial chaos expansion (PCE) for uncertainty propagation, standard and probabilistic Sobol indices to measure global sensitivity analysis, Gaussian process regression with observation-dependent noise structure to construct surrogates based on uncertain data in the space of uncertain parameters, tools for time-series analysis and estimation of time-averaging uncertainties, etc. The methods have been implemented in a non-intrusive way, therefore UQit can be used with any CFD solver conditioned on having appropriate interface. UQ it has been employed for different purposes with the general aim of enhancing our understanding of various sources of uncertainty and influence of different factors in high-fidelity turbulent simulations.

A direction of research has been towards assessing the sensitivity of the scale-resolving simulations of wall-bounded turbulent flows performed by Nek5000, with respect to variation of numerical and modelling parameters. A framework was proposed to assess accuracy, sensitivity and robustness metrics which together can help us draw best-practice guidelines for performing more accurate simulations [24]. The framework was examined for the canonical wall-bounded turbulent flows. The influence of changing the spatial resolution in different directions through systematically varying the elements size and number of collocation (Gauss-Legendre-Lobatto) points per element (equivalent to polynomial order) on the flow quantities of interest (QoIs) has been studied. Moreover, the sensitivity with respect to the variations in the parameters of filtering which is used for numerical stabilization has been extensively investigated. As an additional achievement, the predictive accuracy and sensitivity of Nek5000 have been extensively compared to OpenFOAM employing the above-mentioned UQ-based metrics. Some of the main conclusions of our study [25] can be listed as:

1. For any QoI and at a similar resolution, Nek5000 can be at least twice more accurate than OpenFOAM, see Figure 18.

2. The reduction of the error in QoIs in the space of numerical/computational parameters is non-monotonic which makes posterior error predictions in high-fidelity turbulent flows challenging.

3. The influence of numerical parameters varies between QoIs and also in the wall-normal direction. In particular, the near-wall region in a turbulent boundary layer is where the least influence of the numerics could be observed, see Figure 18.

4. In Nek5000, refinement can be achieved through increasing the number of elements and the polynomial order in each spatial direction and each element. Our observations suggest that if any filtering is involved, a better option is to increase the polynomial order, because there can possibly be an optimal set of filtering parameters which lead to more accurate simulations even at relatively coarse resolutions.

5. For filtering in Nek5000, a high-pass filtering approach with a residual-based source term is preferred to the explicit filtering method.

6. In high-pass filtering, the number of modes which are filtered are more influential than the filter weight.

7. If the guidelines for grid-resolution are derived based on appropriate inner-scaling, then they can be (to a large extent) Reynolds-independent.

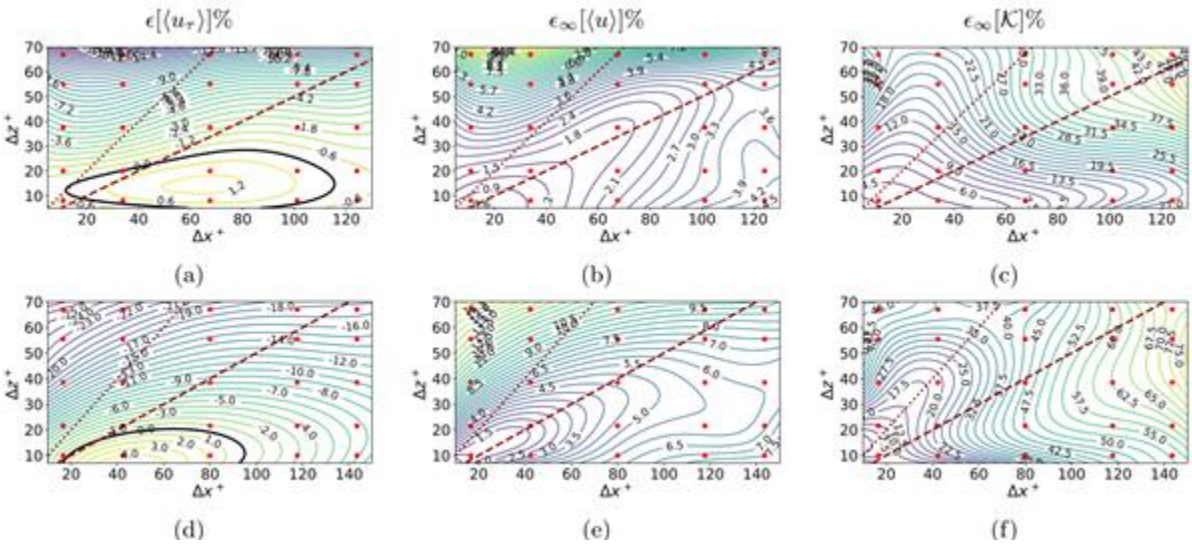

Figure 18: The isolines of the error (in %) in different QoIs of turbulent channel flow in the space of inner-scaled wall parallel grid spacing, taken from [25]. The flow friction Reynolds number is 550 and the simulations are performed by (top) Nek5000 and (bottom) OpenFOAM.

As a part of our efforts, probabilistic versions of PCE and Sobol indices were developed to perform computer experiments based on data which were uncertain due to the finite time-averaging. A very important conclusion was that for a QoI, the highest sensitivity with respect to the variation of numerical parameters is observed at the same physical location where the highest time-averaging uncertainty is estimated. Moreover, the developed technique can be used for estimating error in the space of numerical parameters while confidence intervals for the errors and associated sensitivity indices are also provided.

A main objective of simulation of turbulent flows is to compute turbulence statistics, which can range from low-order moments such as the mean velocity/pressure to higher-order statistics such as the Reynolds stresses. The relevant UQ problem is to accurately estimate the uncertainty in computed turbulence statistics due to the finite averaging time. The main challenge is to

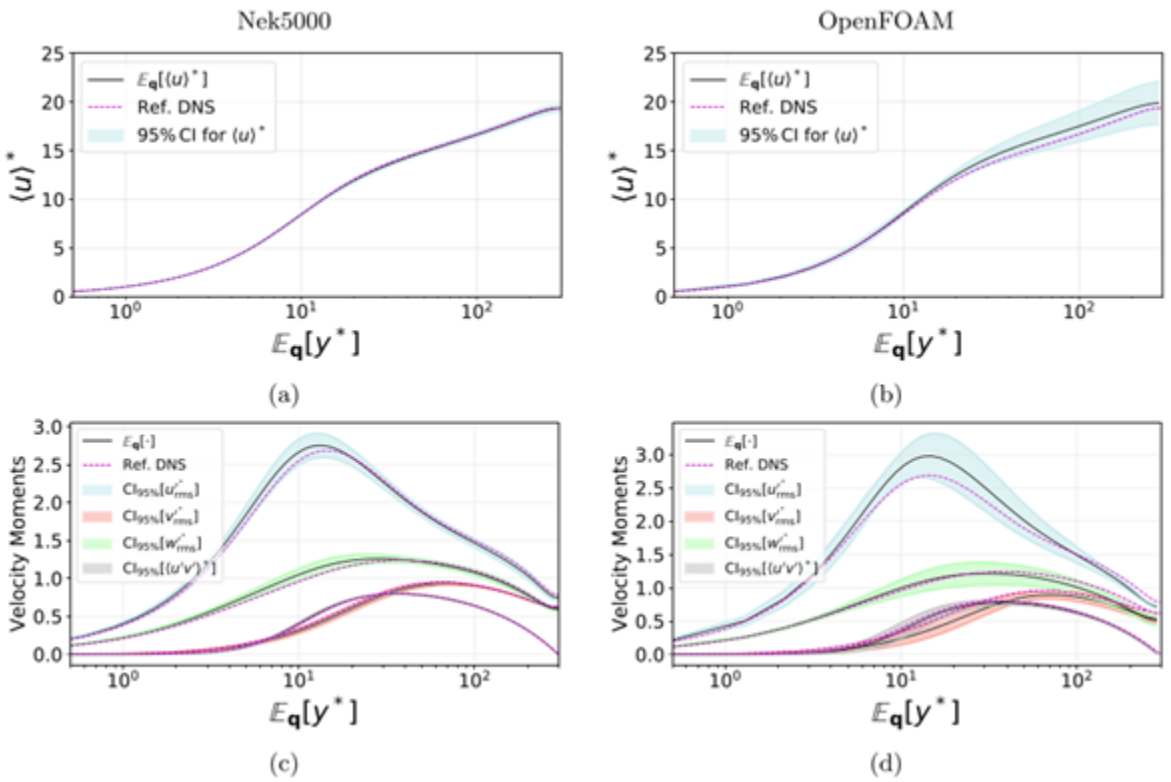

Figure 19: Profiles of the first- and second-order velocity moments of turbulent channel flow at friction Reynolds number equal to 300, taken from [25]. The shaded areas show the 95% confidence intervals due to the variation of grid resolution in the wall-parallel directions.

accurately consider the impact of the autocorrelation and cross-correlation of the time samples of the flow variables which contribute in statistical terms. A part of our efforts within EXCELLERAT has been devoted to further develop techniques for estimating time-averaging uncertainty in turbulent flow statistics as well as providing a set of guidelines for choosing hyper-parameters in various estimation techniques. For the most popular batch-based estimator that is batch-means batch-correlation method [20], we proposed to choose the batch size in such a way that the correlation of the second lag between the batch means becomes close to zero. For the autoregressive-based estimators [21] used for modelling autocorrelation functions, the main hyper-parameter is the order of the autoregressive model which we suggest to be chosen based on the turbulence physical time-scales. For accurate estimation of uncertainty in higher-order statistics and their functionals, we have developed a novel algorithm. As a key feature of the procedure, accurate estimation and inclusion of the cross-covariances between the uncertainty estimators of elementary moments is essential.

6 Workflow Overarching Activities

In addition to the tasks that can directly be assigned to the individual steps of the engineering workflow, performance engineering and efficient implementation play an essential role in the use of HPC, especially when targeting the use of systems that consume several megawatts of electrical energy. Since this is true for all applications executed on large scale HPC resources independent of their position within the engineering workflow, theses general activities and the best practices that were found when dealing with engineering applications are grouped together in the following sections. Besides best practices for node-level and system-level performance engineering in sections 6.1 and 6.3 respectively, the approaches taken when porting engineering applications to new architectures can be found in section 6.2.

6.1 Node Level Performance Engineering

During the span of EXCELLERAT new major architectures appeared challenging the virtual monopoly of Intel that has lasted for ten years. First, the release of the EPYC architecture by AMD was a game changer; as of November 2021 it represents 20% of the top500 share per performance. Also, a lot of movement was registered on Arm based architectures (Huawei, ThunderX, Fujitsu A64FX, AWS graviton, Arm ampere) and is one of the architectures expected for the European processor.

Porting and benchmarking our code in these architectures yielded miscellaneous results. First, the AMD architecture being x86 compliant, portability is straightforward (it even uses the intel compiler if so inclined), but high core count (64) per socket and standard memory bandwidth (204.8 GB/s per socket) make it a challenging architecture for some codes.

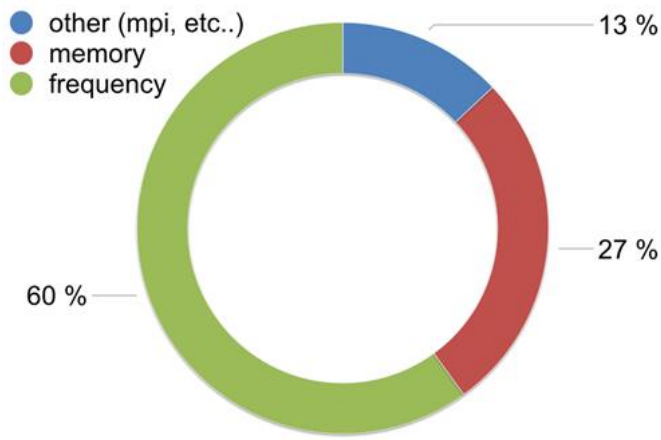

Figure 20: AVBP performance characterisation on AMD Rome processors.

In collaboration with AMD and TGCC, we were able to port and benchmark AVBP in the IRENE Joliot CURIE system (fig. 5).

Arm architecture is trickier and requires specific compilers which can be even specific for a given processor type (see A64FX). For our first experience we were fortunate to get access to multiple systems and mostly used the gnu compiler (version 10 or 11) as they are already known to the code developers and also offer support for Arm instructions.

Portability using the gnu compiler revealed some surprises as some Fortran or C constructs although accepted by intel compilers are not strictly compliant to the norm but overall, no problem was detected. However, performance is very different depending on the flavour of Arm tested. Only recent systems integrate the vectorisation SVE options and when they do it is not

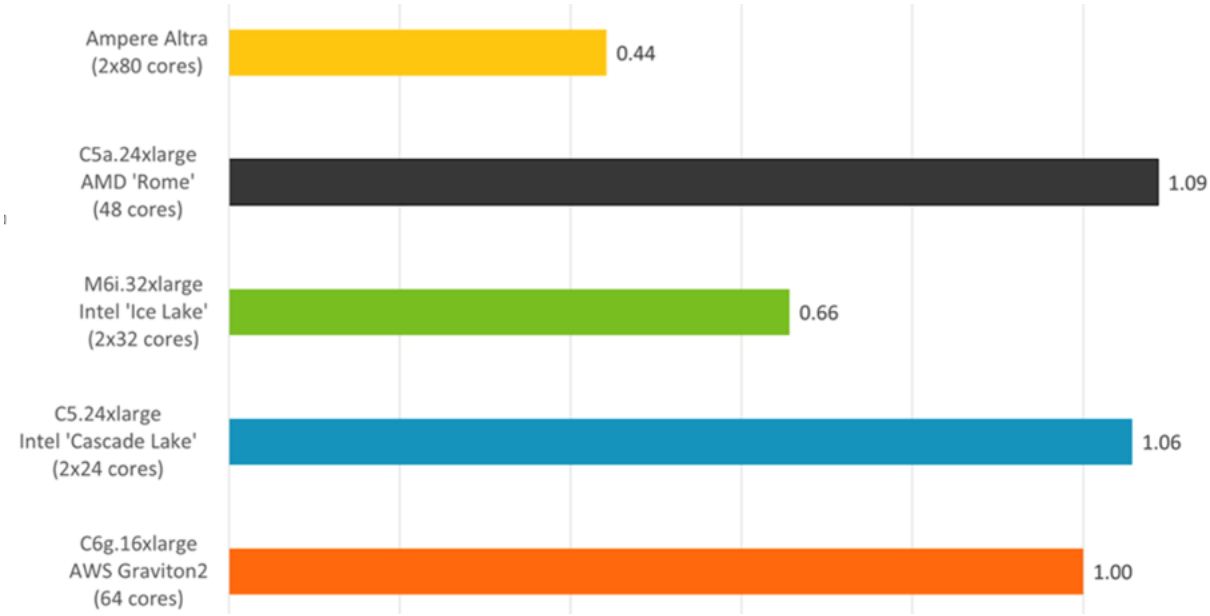

Figure 21: normalized timing (lower is better) for 20M simulation using AVBP using AWS resources.

easy to vectorize as the compiler is not always able too. Figure 21 compares the most promising architectures with Rome and Intel processors on a real simulation using AVBP using a full single node. Out of the box, using the gnu compiler is fully functional and provides very acceptable performance. More work is expected in the future as performance analysis tools catch up to the new architectures and provide more insight into bottlenecks for vectorisation and data locality errors.

Co-execution (BSC)

We can affirm with quite a certainty that future exascale systems will be heterogeneous, including nodes with accelerators such as GPUs. As the size of the systems grows, and thus more nodes are engaged in a single simulation, we can also expect higher variability on the performance among the computing devices engaged in a simulation. Apart from this explosion of the parallelism, there are technical aspects related with variability, such as the hardware-enforced mechanisms to preserve the thermal design limits. In this context, dynamic load balancing (DLB) becomes a must for the parallel efficiency of any simulation code.

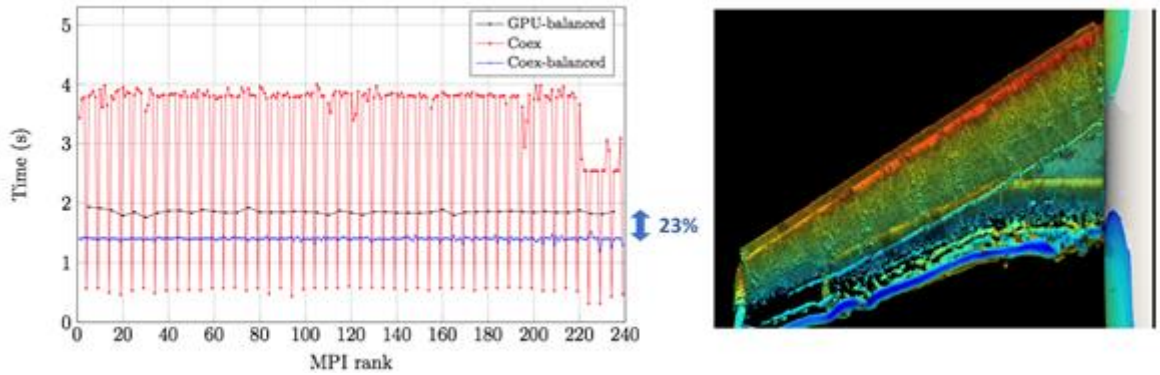

Figure 22: (Left): Comparison of (balanced) co-execution vs pure GPU execution – elapsed time per MPI Rank. (Right): Snapshot of Q iso-surfaces of the turbulent flow around an airplane.

In the first year of the EXCELLERAT project, Alya was provisioned with a distributed memory DLB mechanism, complementary to the node-level load balancing mechanisms already in place. The kernel parts of the method are an efficient in-house SFC-based mesh practitioner, and an online redistribution module to migrate the simulation between two different partitions. The availability of those capabilities allows to adjust the partition according to runtime measurements.

We have focused on maximizing the parallel performance of the mesh partition process to minimize the overhead of the load balancing, to achieve this goal a massively parallel SFC based mesh partitioner was implemented [3]. We then applied all this technology to perform simulations on the heterogeneous POWER9 cluster installed at the Barcelona Supercomputing Centre, with an architecture very similar to that of the Summit supercomputer from the Oak Ridge National Laboratory – ranked first in the top500 list at that time. In the BSC POWER9 cluster, which has 4 NVIDIA P100 GPUS per node, we demonstrated that we could perform a well-balanced co-execution using both the CPUs and GPUs simultaneously, being that 23% faster than using only the GPUs. In practice, this represents a performance boost equivalent to attaching an additional GPU per node. This research was published at the Future Generation Computer Systems journal [26] where a full analysis of the code performance is given. Sample results of the elapsed time per MPI Rank are given in Figure 22 (left), while a snapshot of Q-vorticity along the wing is shown in Figure 22 (right).

6.2 Porting to New Architecture

When porting to new architectures, especially ones which are novel for HPC, it is important to ask two questions. Firstly, what is the purpose of doing this port (i.e. what properties will this architecture give me over and above the hardware currently in use), and secondly what code level modifications are required to most effectively exploit this new architecture. In EXCELLERAT there have been numerous successes in porting our applications to numerous new architectures, including GPUs and FPGAs, and these two architectures demonstrate these points well.

When porting from CPUs to GPUs the major benefit one will obtain will be that of computational performance. Put simply, GPUs are engines for floating point arithmetic and therefore if one's code is bound by a lack of computation on the CPU then they are a good technology to consider. Furthermore, GPUs tend to have excellent memory bandwidth and-so can often effectively keep the compute units (the symmetric multiprocessors) fed with data sufficiently well. FPGAs by comparison are suited to other code properties, typically where code on the CPU is bound by memory accesses or other core issues. It is true that GPUs have excellent memory bandwidth, but if one's code is bound by memory latency (often due to a challenging memory access pattern such as many indirect memory accesses) then the tailoring of the electronics to the application in question can provide significant benefits as we have seen during the CoE with Nekbone and Alya codes. Whilst the exact choice of profiling tool can be personal preference to some extent, technologies such as Intel's Vtune can be highly effective in highlighting code bound issues and consequently detailed profiling on current architectures should always be a first step.

GPUs are organised around the principal of vectorisation, whereas FPGAs are organised around dataflow. This can mean that when porting to such architectures one must fully embrace these paradigm shifts and enhance their code to match. This can result in very significant algorithmic level changes, but these are necessary if one is to achieve best performance. Throughout the

CoE we have explored in depth techniques to port codes to GPUs and FPGAs, with numerous lessons learnt and successes achieved. However, it is fair to say that the devil is in the detail and one of the challenges with the tooling is that it is possible to get code running fairly quickly on these architectures. Whilst this sounds positive, in reality the performance that one obtains often leaves a lot to be desired and then significant expertise and insight is required to tune up the applications to the specific hardware. It is critically important to be aware of the latest techniques for the architecture that one is targeting, whilst this sounds obvious for more novel hardware such as FPGAs there is a wealth of experience being developed and disseminated. Consequently, it can be a challenge to maintain one's knowledge and ensure that the latest approaches are being used.

The work done in this CoE exploring the acceleration of NekBone on FPGAs is a good example of this last point, where approximately a 5000 times difference in performance existed between the initial port onto FPGAs and the finalised optimised kernel. Similarly, for the GPU acceleration of Nek5000 and NekBone, the initial GPU port, where the initial OpenACC implementation was relatively simple to implement but had significant performance problems. One of the main issues was that as Nek5000 is written as a set of internal maths libraries which are then repeatedly, often in loops simply adding OpenACC directives to the maths functions results in many small inefficient kernels called sequentially, which was inefficient, particularly in the case where the calculation is followed by a reduction. Manually merging these kernels and then writing the reduction by hand implementation significantly improved the overall performance of those kernels. It is only when one is drawing in on the top-level performance that they are beating other technologies, but the danger is that developers observe the initial performance and then decide it is not worth exploring the technology further.

Another question developers face is the choice of technology to do the porting. This is specially the case today for GPUs if you have a legacy code. 3 main technologies exist today to port/develop codes on GPUs with variable pros and cons:

- Pragma based acceleration with OpenACC/OpenMP
- Cuda Library
- Domain Specific Languages / Code generation.

The main divergence between these technologies is the return on investment and the time involved in development and maintenance and differ greatly on whether you are working on a legacy code or a new solver.

Pragma based programming via OpenMP and OpenACC offers best return on investment for legacy codes that cannot invest on a full code rewrite as evidence by the work performance on EXCELLERAT in the AVBP application (350k, 30-year-old legacy code in Fortran) where an acceleration of factor 5 full CPU node versus 4 A100 Nvidia GPUs is observer on the Juwels Booster nodes at JSC and scales up to 1000 CPUs. Some caveats apply though. Pragmas rely heavily on the compiler implementation and from one version to another fringe behaviour can be expected. Also, not all compilers offer the same level of implementation: OpenACC is mostly limited to the Nvidia SDK, OpenMP 5 implementation greatly differs today.

Additionally, pragma-based programming offers the possibility to run the code without acceleration naturally but also makes it easier for developers to ignore them rendering maintenance of the code all the more critical.

Cuda library support offers the best performance per investment ratio but often require the recoding of the whole application making them more viable for new projects or full

refactorings. Moreover, each new release of CUDA can require a new porting/development phase to adapt to the new version.

Finally, domain specific languages (DSL) and code generation can be perceived as the best tools for new codes today. Indeed, these methods offer the promise of minimising code refactoring / porting for future generations of developers by abstracting code operators to their numerical/physical representations. The burden of supporting the architectures is offloaded completely to the DSL's developers. Of course, such an approach requires full application re-write. Kokkos can be seen as a DSL as well although a lower level one as it focuses on memory mapping and access and not on physical operators.

Nevertheless, all of these approaches have seen considerable success stories and are expected to be the key for exascale computing.

6.3 System-Level Performance Engineering

The property of strong-scaling (reducing the time-to-solution when the computing resources are increased), is a primary goal on the development of CFD codes, and particularly for Alya. A code with good strong scaling will allow to reduce the simulation time when there are more resources available.

In EXCELLERAT the strong scaling was tested for the use-case U1C2 that consists of a multiphase reacting flow field of a double-swirl air blast spray flame of a test rig being constructed at TU Berlin. The performance of a Eulerian-Lagrangian framework, where the disperse phase is represented by Lagrangian droplets and the gas phase is described by the Eulerian phase, was analysed. The mesh consisted of 1 billion cells and about 200k Lagrangian particles. The tests aimed to show the acceleration achieved from using 100 up to 400 nodes of the MareNostrum IV supercomputer. The parallel efficiency (PE) achieved was 91%. The MareNostrum nodes are composed of 48 CPU-cores, therefore., the maximum number of CPU-cores considered was 19200. Note that this is a multi-physics case that includes many physical

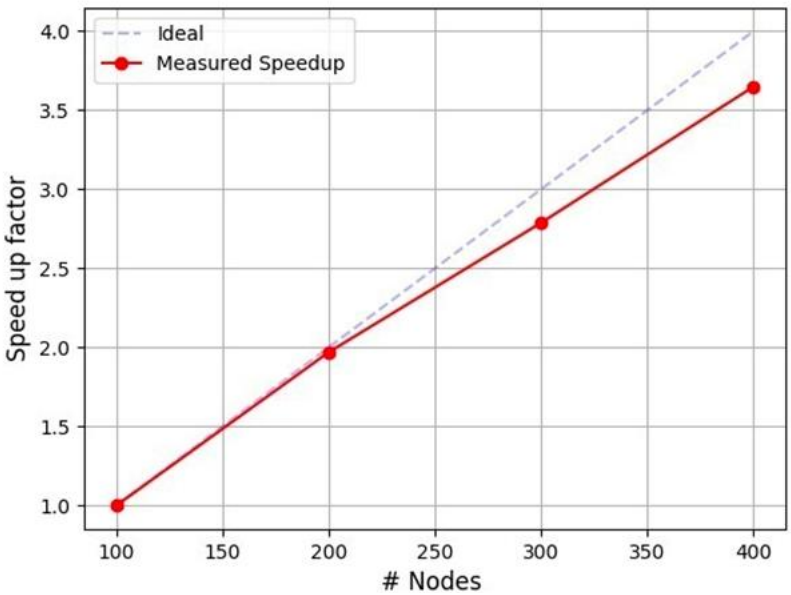

Figure 23: Strong scaling performance of the Alya code on the MareNostrum IV supercomputer for the case U1C2.

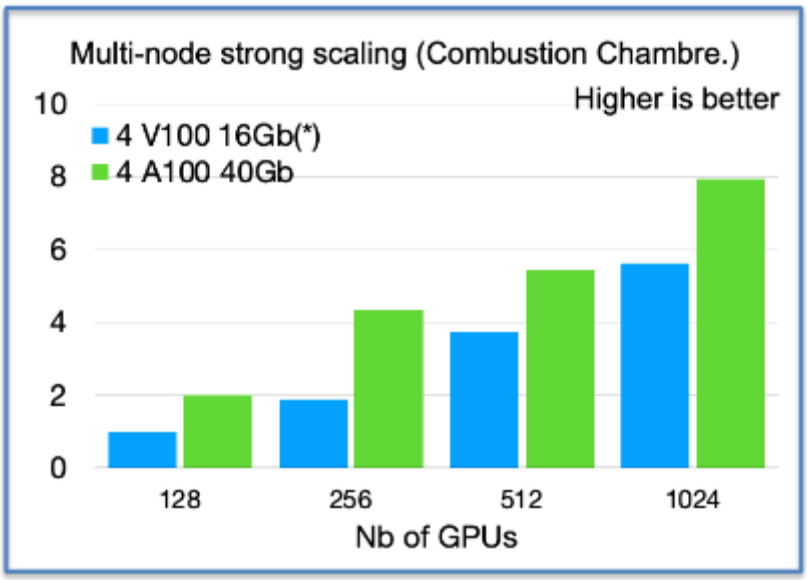

Figure 24: Strong scaling on U2C3 for AVBP on JUWELS BOOSER (A100) and JEANZAY (V100).

phenomena: the particle transport, heating and evaporation, the Navier-Stokes equations, the energy equations, and the transport equations for the controlling variables used in the flamelet method.
The strong scaling results are shown in Figure 23, the resources used are multiplied by four and the acceleration achieved is 3.6. The main degradation factor in these cases is the communications overhead.

Strong scaling using GPU acceleration is even more challenging and depends dually on the hardware and software implementation on the cluster in parallel to the application. The key to strong scaling using GPUs is GPU direct support, GPU direct provides direct communication between NVIDIA GPUs in remote systems avoiding data copies between the GPU and CPU. On JUWELS Booster, on 4 nodes, an improvement of 20% in time to solution is observed when enabling GPU direct communication compared to default copy/ CPU MPI communication.

7 References

[1] C. Geuzaine and J.-F. Remacle, "Gmsh," 29 November 2019. [Online]. Available: http://gmsh.info/.

[2] "OPEN CASCADE SAS," 29 November 2019. [Online]. Available: https://dev.opencascade.org/.

[3] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing,* vol. 20, p. 359–392, January 1998.

[4] C. Chevalier and F. Pellegrini, "PT-Scotch: A tool for efficient parallel graph ordering," *Parallel Computing,* vol. 34, p. 318–331, July 2008.

[5] K. Devine, E. Boman, R. Heaphy, B. Hendrickson and C. Vaughan, "Zoltan data management services for parallel dynamic applications," *Computing in Science {\&}amp$\mathsemicolon$ Engineering,* vol. 4, p. 90–96, 2002.

[6] R. Borrell, J. C. Cajas, D. Mira, A. Taha, S. Koric, M. Vázquez and G. Houzeaux, "Parallel mesh partitioning based on space filling curves," *Computers {\&}amp$\mathsemicolon$ Fluids,* vol. 173, p. 264–272, September 2018.

[7] R. Borrell, G. Oyarzun, D. Dosimont and G. Houzeaux, *Parallel SFC-based mesh partitioning and load balancing,* arXiv, 2020.

[8] E. Chow, "A Priori Sparsity Patterns for Parallel Sparse Approximate Inverse Preconditioners," *SIAM Journal on Scientific Computing,* vol. 21, p. 1804–1822, January 2000.

[9] E. Chow, "Parallel Implementation and Practical Use of Sparse Approximate Inverse Preconditioners with a Priori Sparsity Patterns," *The International Journal of High Performance Computing Applications,* vol. 15, p. 56–74, February 2001.

[10] S. Laut, R. Borrell and M. Casas, "Cache-aware Sparse Patterns for the Factorized Sparse Approximate Inverse Preconditioner," in *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing,* 2021.

[11] C. Burstedde, L. C. Wilcox and O. Ghattas, "p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees," *SIAM Journal on Scientific Computing,* vol. 33, p. 1103–1133, January 2011.

[12] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker and C. S. Woodward, "SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers," *ACM Transactions on Mathematical Software,* vol. 31, p. 363–396, September 2005.

[13] "DLB Library," 27 April 2022. [Online]. Available: https://www.bsc.es/research-and-development/software-and-apps/software-list/dlb-library-dynamic-load-balancing.

[14] J. Duda, "Asymmetric numeral systems:," [Online]. Available: https://arxiv.org/pdf/1311.2540.pdf. [Accessed 28 April 2022].

[15] "Zstandard," [Online]. Available: https://facebook.github.io/zstd/. [Accessed 2021 November 2021].

[16] "BigWhoop," [Online]. Available: https://projects.hlrs.de/projects/bwc/. [Accessed 28 April 2022].

[17] C. Wenzel, P. Vogler, J. Peter, U. Rist and K. Markus, "Application of a JPEG 2000-based data compression algorithm to DNS of compressible turbulent boundary layers up toReθ=6600," 2020.

[18] P. Lindstrom, "Fixed-Rate Compressed Floating-Point Arrays," *IEEE Transactions on Visualization and Computer Graphics,* vol. 20, pp. 2674-2683, 2014.

[19] A. Loddoch and J. Schmalzl, "Variable quality compression of fluid dynamical data sets using a 3-D DCT technique: DATA COMPRESSION," *Geochemistry, Geophysics, Geosystems,* vol. 70, 2006.

[20] S. Russo and P. Luchini, "A fast algorithm for the estimation of statistical error in DNS (or experimental) time averages," *Journal of Computational Physics,* vol. 347, p. 328–340, October 2017.

[21] T. A. Oliver, N. Malaya, R. Ulerich and R. D. Moser, "Estimating uncertainties in statistics computed from direct numerical simulation," *Physics of Fluids,* vol. 26, p. 035101, March 2014.

[22] C. Gscheidle, S. Rezaeiravesh, J. Garcke and P. Schlatter, "In-situ estimation of time-averaging unvertainties in turbulent flow simulations," in *Workshop "Multi-Scale, Multi-physics and Coupled Problems on highly parallel systems (MMPC)",* Kobe, 2022.

[23] S. Rezaeiravesh, R. Vinuesa and P. Schlatter, "UQit: A Python package for uncertainty quantification (UQ) in computational fluid dynamics (CFD)," *Journal of Open Source Software,* vol. 6, p. 2871, April 2021.

[24] S. Rezaeiravesh, R. Vinuesa and P. Schlatter, *An Uncertainty-Quantification Framework for Assessing Accuracy, Sensitivity, and Robustness in Computational Fluid Dynamics,* arXiv, 2020.

[25] S. Rezaeiravesh, R. Vinuesa and P. Schlatter, "On numerical uncertainties in scale-resolving simulations of canonical wall turbulence," *Computers {\&}amp$\mathsemicolon$ Fluids,* vol. 227, p. 105024, September 2021.

[26] R. Borrell, D. Dosimont, M. Garcia-Gasulla, G. Houzeaux, O. Lehmkuhl, V. Mehta, H. Owen, M. Vázquez and G. Oyarzun, "Heterogeneous CPU/GPU co-execution of CFD simulations on the POWER9 architecture: Application to airplane aerodynamics," *Future Generation Computer Systems,* vol. 107, p. 31–48, June 2020.

[27] S. R. J. G. a. P. S. C. Gscheidle, *In-situ estimation of time-averaging uncertainties in turbulent flow simulations,* Kobe, 2022.