HORIZON-EUROHPC-JU-2021-COE-01

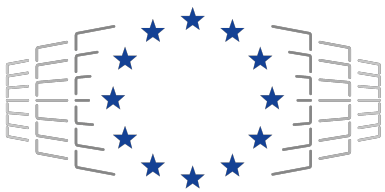
Centre of Excellence in Exascale CFD

CEEC – Centre of Excellence in Exascale CFD

Grant Agreement Number: 101093393

D3.1 – Analysis of the CEEC codes and underlying solvers: Requirements and strategies definition

WP3: Exascale algorithms



EuroHPC Joint Undertaking

Document Information

Deliverable Number	D3.1
Deliverable Name	Analysis of the CEEC codes and underlying solvers: Requirements and strategies definition
Due Date	31/08/2023 (PM 08)
Deliverable lead	UmU
Authors	Roman Iakymchuk (UmU), Manuel Muensch (FAU), Martin Berggren (UmU), Niels Aage (DTU), Anna Schwarz (USTUTT), Daniel Kempf (USTUTT), Harald Köster (FAU), Guillaume Houzeaux (BSC), Ananias Tomboulides (AUTH), Timofey Mukha (KTH)
Responsible Author	Roman Iakymchuk (UmU) riakymch@cs.umu.se
Keywords	consortium codes, solvers, requirements analysis, strategies definition
WP	WP3
Nature	R
Dissemination Level	PU
Final Version Date	31/08/2023
Reviewed by	Niclas Jansson (KTH), Filippo Mantovani (BSC)
MGT Board Approval	31/08/2023

Acknowledgment:

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Sweden, Germany, Spain, Greece, and Denmark under grant agreement No 101093393.

Document History

Partner	Date	Comment	Version
UmU, DTU, USTUTT	09/04/23	Define questionnaire for the consortium codes	0.1
UmU, KTH, FAU	16/05/23	Initial analysis of the questionnaire outputs	0.11
UmU	16/06/03	Preliminary version with all codes	0.2
UmU	30/06/23	Strategies for scalable, fast, and efficient solvers	0.3
FAU	03/07/23	Adaptivity and fault tolerance strategies	0.4
UmU, DTU	07/07/23	Topology optimization strategies	0.5
USTUTT, AUTH, KTH, BSC	10/07/23	Revision and updates by code owners	0.6
UmU	18/07/23	Version ready for internal review	0.7
UmU, USTUTT, DTU, FAU	24/08/23	Incorporated reviewers' comments	0.8
UmU	29/08/23	Final version	1.0

Executive Summary

This document is the first deliverable of work package 3 – 'Exascale Algorithms' of the EuroHPC JU Center of Excellence in Exascale CFD (CEEC). This work package is concerned with all efforts required for improving algorithms in the CEEC codes for efficient exploitation of Exascale architectures. In particular, we focus on improving scalability and numerical properties of cornerstone algorithms in computational fluid dynamics (CFD) codes; providing mixed-precision algorithmic solutions for better energy footprint and/ or faster execution; enabling error control and assure robustness of algorithms; facilitating exascale design optimization.

This first deliverable 'D3.1 – Analysis of the CEEC codes and underlying solvers: Requirements and strategies definition' provides analysis of consortium codes and their underlying solvers and summarizes the outcome of the work performed within Tasks 3.1, 3.2, 3.3, 3.4, and 3.5 between Month 1 and 8 of the project. The document also provides the requirements and strategies definition to reach the project goals. The other deliverables in work package 3 will be focused on the actual implementation of the defined strategies and approaches, as well as their integration into the consortium codes.

Contents

1 Introduction

The Center of Excellence in Exascale CFD (CEEC) implements exascale ready workflows for addressing relevant challenges for future exascale systems, including those procured by EuroHPC. The significant improvement in energy efficiency will be facilitated through efficient exploitation of accelerated hardware architectures (GPUs) and novel adaptive mixed-precision calculations. Emphasis is furthermore given to new or improved algorithms that are needed to exploit upcoming exascale architectures. The efforts of the center are driven by a collection of six different lighthouse cases of physical and engineering interest, ranging from aeronautical to atmospheric flows.

In order to establish a roadmap for reaching the project goals driven by the light-house cases, for more details we refer to [7], we conduct analysis of the consortium codes and their underlying solvers in the form of questionnaire with the engagement and outputs from the code owners. As a result of this questionnaire and internal discussions, we draw requirements for each code and outline potential algorithmic development strategies in order to meet the work package 3 objectives.

The document is organised as follows: Section 2 presents description of the consortium codes and both the requirements and potential strategies toward reaching the project goals. Section 3 summarizes the strategies and approaches toward reliable, efficient, and fast algorithmic solvers.

2 Analysis of the consortium codes

Together with work package 1 – 'Exascale light-house cases', we in work package 3 set up a questionnaire in order to gather more detailed information on the codes, their solvers, and to derive requirements for algorithmic developments. Thus, the questionnaire poses the following main questions:

- How does the solver work on matrices or matrix-free? This topic is related to the underlying algorithmic solvers (implicit, explicit, Krylov type, Runge-Kutta, etc) and preconditioning techniques.
- Does the application offer a proxy-app and/ or a (simplified) test case? This topic is useful for preliminary studies of required computing precision and also on novel architectures where the whole software stack may not be ready for supporting the execution of the whole application.
- Does the application rely upon adjoint solver, algorithmic/ automatic differentiation, etc.? This topic is related to topology optimization and targets some optimization-specific issues.
- Does the application foresee the use of check-pointing mechanisms? This is useful for studying fault tolerance aspects within the application and when deploying on new systems (such as the new EuroHPC ones).
- Does the code owner already consider any potential strategies for enhancing scalability, numerical properties, and energy efficiency? This topic is relevant to the algorithmic development work, covering mixed precision solvers.

In addition, we provide a brief introductory information on each consortium code. Further details on the codes with their requirements justified by the lighthouse case are given in

'D1.1 – CEEC exascale lighthouse cases and their needs' due on PM8 [7].

2.1 FLEXI

2.1.1 Description

FLEXI is a high-order accurate, open source solver for general partial differential equations of hyperbolic/parabolic-type based on the discontinuous variant of the spectral element method, the discontinuous Galerkin (DG) spectral element method (DGSEM) [12]. This enables an element-local and efficient scheme, even on highly parallel systems. FLEXI is written in modern Fortran, for the parallelization it uses MPI and for parallel I/O the HDF5 library. Additionally, FLEXI requires the math library LAPACK and for optional post-processing the FFTW library. FLEXI's primary area of application are direct numerical simulation (DNS) and large eddy simulations (LES) of multiscale- and multi-physics problems, where the fluid phase is governed by the compressible Navier–Stokes–Fourier equations (NSE), which includes turbulent flows and shock-turbulence interaction. Since high-order schemes are subject to oscillations at discontinuities, additional stabilization techniques are necessary to accurately predict strong compression shock waves across which the flow properties change drastically. In FLEXI, discontinuities in the solution such as shock waves are handled by shock-capturing strategies through an elementwise h-refinement strategy in a localized and stable manner. This approach is based on a finite volume (FV) sub-cell approach, where troubled cells are switched from a DG to a FV discretization. Another, more stable approach relies on a convex blending of the low-order FV sub-cell with a high-order DG scheme. To ensure nonlinear stability in underresolved flow regions as occurring in a LES, FLEXI uses kinetic energy or entropy stable discretizations based on the split form of the advective terms. Various closure strategies for the NSE in a LES formulation based on implicit or explicit modeling are available. High temporal accuracy is ensured through high-order, low-storage explicit Runge-Kutta (RK) schemes. FLEXI supports unstructured grids of tensor-product elements due to the DGSEM and high-order accurate geometry representation including curved grid cells and hanging nodes. Grid adaptation is handled through a conservative mortar interface definition.

2.1.2 Requirements

Underlying algorithmic solvers and (simplified) test cases While there is no dedicated proxy-application for FLEXI, there is a possibility to reduce its compilation; so that, we can test the relevant functionalities. FLEXI uses preprocessor flags to exclude certain features, e.g. the parabolic part of the Navier–Stokes equations, switch between 2D /3D computation and the usage of an additional FV operator for shock capturing, which results in a reduced code with specific functionalities tailored to cover the relevant investigations and applications within the CEEC project. This limited functionality of FLEXI will be useful during the initial exploration of the solvers and their potential alternatives. A few small test cases are available for the reduced version of the code on the public GitHub repository of FLEXI[1], otherwise these are provided. Such test cases should cover the relevant functionalities of the code for the CEEC project.

For the mixed-precision investigations, it is important to note that the underlying solver of FLEXI discretizes the NSE with the DGSEM and the solution is forwarded in time

[1] `https://github.com/flexi-framework/flexi`

by an explicit, low-storage RK scheme of arbitrary order. Since FLEXI relies on an explicit time-integration scheme, it is matrix-free and does not require to allocate, store, and distribute large global matrices. Moreover, rounding errors tend to accumulate and affect especially later stages of the computations. Thus, a potential advantage of precision cropping could be achieved, e.g., by reducing the precision in the shock-capturing scheme, here the FV sub-cell discretization. This is especially suitable if the convex blending scheme is employed, where rounding errors due to the precision cropping of the FV sub-cell scheme will affect the accuracy of the solution not significantly due to the smooth blending of the low-order with the high-order DG method.

The FLEXI team is not considering any alternatives to the compressible NSE discretized in time and space by an explicit RK schemes and a high-order, hybrid DG/FV sub-cell method, respectively.

Fault tolerance In regular time intervals, a solution state is written in parallel in an HDF5-file which allows a restart or resumption of the simulation. The solution state consists of the solution vector, containing the density, momentum and total energy density, for each degree of freedom. Check-pointing in the context of adjoint is not relevant to FLEXI.

Explicit fault tolerance mechanisms have not been implemented and are not the subject of further development of FLEXI within CEEC. However, there is a possibility to define a threshold for the performance index, which is the time required to advance a single degree of freedom from one stage of the Runge-Kutta time integration scheme to the next, which is evaluated at regular intervals to judge the performance of the simulation (it should be a constant ideally). If the threshold is exceeded, the simulation will be terminated to prevent unnecessary resource consumption. A sudden rise of the performance index is typically associated with a faulty compute node or network switch.

2.1.3 Potential strategies

Here, we outline potential strategies for enhancing scalability, numerical properties, energy efficiency, as well as execution time. Scalability of the current version of FLEXI is excellent on CPU-based HPC clusters [5]. Within CEEC, FLEXI will be transitioned to GPU-based systems to achieve similar scalability applying the same communication latency hiding strategy as described in [12]. With respect to the numerical properties of FLEXI, the considered LHC consists of strong shock waves which require special treatment to stabilize the DG scheme. The shock capturing technique based on a convex blending approach of a DG with a FV scheme demonstrates the potential of such a scheme by providing robustness while maintaining the accuracy of the solution in smooth regions. To gain an optimal balance between robustness and accuracy, further improvements and enhancements of this approach will be tackled within the CEEC project. This could improve the energy-to-solution properties. To compute the LHC with FLEXI, wall modelling is utilized to reduce the number of required degrees of freedom and to increase the time step, resulting in a reduced execution time and possibly better energy-to-solution properties.

We explore the possibility of adapting/cropping the working precision of the FLEXI code on few examples. Since the scheme is explicit, rounding errors have a tendency to accumulate and have a larger impact, especially on the later stages of computations. Hence, a feasible scenario is to lower the precision of the FV sub-cell operator.

2.2 Alya

2.2.1 Description

Alya[2] [18] is a multi-physics simulation code developed in modern Fortran at Barcelona Supercomputing Center (BSC). From its inception, Alya code is designed using advanced High-Performance Computing programming techniques to solve coupled problems on supercomputers efficiently, especially taking into account heterogeneous architectures. The target domain is engineering, with all its particular features: complex geometries and unstructured meshes, coupled multi-physics with exotic coupling schemes and physical models, ill-posed problems, flexibility needs for rapidly including new models, etc. Since its beginnings in 2004, Alya shows good scalability results when solving single-physics problems such as fluid mechanics, solid mechanics, heat transfer, combustion, etc. Over time, we have made a concerted effort to maintain and even improve scalability for multi-physics problems. This poses challenges on multiple fronts, including numerical models, parallel implementation, physical coupling models, algorithms and solution schemes, meshing processes, etc. Alya is present in the PRACE benchmark suite (UEABS) together with Code_Saturne in the field of Computational Fluid Dynamics and, thus, is considered as the reference in the EU framework for supercomputing. The core Alya Dev Team has today around 50 members, distributed among the BSC and its spinoff, ELEM Biotech.

2.2.2 Requirements

Underlying algorithmic solvers and (simplified) test cases The open-alya version is a light version of Alya. There is also a miniapp (from the EuroHPC JU CoE PoP project) that performs the Navier-Stokes assembly of an explicit solver.

To study the underlying algorithmic solutions, we will work on a few small cases first. Such small testcases will be provided and they cover the relevant functionalities of Alya via its miniapp.

Alya supports both explicit and implicit solvers. For the explicit solvers, the Runge-Kutta family is considered, especially Runge-Kutta scheme of a fourth order. These solvers are matrix-free. For the implicit solvers, Krylov-type solvers and coarse solvers (deflation) are used. The implict solvers are complemented with preconditioning techniques such as Linelet, RAS, and approximate inverse. The implicit solvers require to construct a global matrix. Such matrices can be exported to matrix market format for further inspection in Matlab; several matrix storage files are produced when done in parallel. Alya offers a possibility to substitute these solvers and preconditioners, which offers a potential to facilitate the algorithmic development.

Concerning alternative algorithmic solutions, Alya provides interfaces to work with algorithmic implementations from PETSC, Maphys, and MUMPS libraries. Hence, plugging our algorithmic solutions, which can be developed separately, is a possibility.

Fault tolerance For check-pointing, we write a restart file using parallel MPI I/O according to the user specifications. We have the option to write this file in a partition-independent way to restart with a different number of MPI tasks.

[2] `https://gitlab.com/bsc-alya/alya`

2.2.3 Potential strategies

We outline potential strategies for enhancing scalability, numerical properties, energy efficiency, as well as execution time. We consider to explore a possibility to crop precision in the Alya solvers with the help of the mixed-precision approach reinforced by the iterative refinement. With the explicit scheme, the rounding errors have a tendency to accumulate and have a larger impact, especially on the later stages of computations. The idea is to commence with lower precision and increase the precision later at the higher iterations count.

For better resources utilization as well as to adapt to the heterogeneous environments as on the modern HPC systems, we intend to enhance the work on load balancing (now done with the DLB library) and vectorization. Exploring other matrix storage formats is also foreseen for the purpose of reducing the data movement, e.g. for sparse matrices. Porting to GPUs is a part of the CEEC involvement of the Alya's team.

From the user feedback, the check-pointing strategy we are following seems to be sufficient. But, any smart fault-tolerance solution is welcome. We had the possibility to use FTI for fault tolerance, but this is no longer maintained.

2.3 Neko

2.3.1 Description

Neko [10] is a portable framework for high-order spectral element based simulations on hexahedral meshes, mainly focusing on incompressible flow simulations. Neko has its roots in the spectral element code Nek5000 from UChicago/ ANL, from where many of the namings, code structure and numerical methods are adopted. The framework is written in modern Fortran and adopts an object-oriented approach, allowing for multi-tier abstractions of the solver stack and facilitating various hardware backends, ranging from general-purpose processors, accelerators to vector processors, and as well as limited FPGA support. Neko focuses on single core/ single accelerator efficiency via fast tensor product operator evaluations. For high-order methods, assembling either the local element matrix or the full stiffness matrix is prohibitively expensive. Therefore, a key to achieving good performance in spectral element methods is to consider a matrix-free formulation, where one always works with the unassembled matrix on a per-element basis. Gather–scatter operations are used to ensure continuity of functions on the element level, operating on both intra-node and inter-node element data. Currently, Neko uses MPI for inter-node parallelism and parallel I/O for production runs. However, one-sided options such as Coarray Fortran based gather-scatter kernels are under development.

The primary consideration in Neko is how to efficiently utilize different computer backends without re-implementing the whole framework for each backend while maintaining the core of the solver in modern Fortran. We solve this problem by considering the weak form of the equations used in the spectral element method. The weak formulation allows us to formulate equations as abstract problems which enable us to keep the abstractions at the top level of the software stack and reduce the amount of platform-dependent kernels to a minimum. The multi-tier abstractions in Neko are realized using abstract Fortran types. These abstract types describe a flow solver's common parts, from how to compute a time-step in a solver, function spaces and various fields, as well as matrix-vector and gather-scatter kernels. Furthermore, these abstract types are associated with an actual implementation in an extended derived type, allowing for hardware or programming model

specific implementations, all interchangeable at runtime. This way, Neko can accommodate different backends with both native and offloading type execution models without unnecessary code duplication in the solver stack.

2.3.2 Requirements

Underlying algorithmic solvers and (simplified) test cases Proxy-applications are available for key kernels, however they are not representative for large cases. Due to that, it is better to use available scaled down problems as for FLEXI. Few simplified test cases (preferably with the runtime within several minutes to begin with) are available on the Neko public GitHub repository[3].

The underlying solvers are matrix-free Krylov solvers: Generalized minimal residual method (GMRES), Conjugate Gradient (CG), and pipelined CG (pipe-CG). These solvers use various preconditioners: block Jacobi and hybrid additive Schwarz. The solvers are coupled within the code. While mixed-precision versions of the solvers or their alternatives can be tested outside, it is still easy to change between solvers that follows the matrix-free SEM approach. Furthermore, the solvers are matrix free; there is no straightforward possibility to form a matrix for testing outside the code.

Fault tolerance Neko relies on a periodic check-pointing mechanism as the way to enable fault-resilient executions.

Concerning additional fault-tolerance mechanisms, fault tolerance (e.g. like in ExaFLOW[4] by replaying the initialization) would be interesting to consider, or a way to gracefully check-point after a failure, e.g. using a RAID-type of reconstruction.

Topology optimization Neko does not have an adjoint solver, however, this facility will be implemented within the scope of CEEC. The same applies to the thermal transport equation that needs to be solved as a one-way coupled problem and taken into account in the sensitivity analysis. The code relies upon MPI I/O to exchange information in massively parallel settings.

2.3.3 Potential strategies

Here, we outline potential strategies for enhancing scalability, numerical properties, energy footprint, as well as execution time. We consider task-parallel preconditioners to ensure high GPU utilization. The paper [2] on the Conjugated Gradient method preconditioned with Block-Jacobi suggests that lowering precision in the preconditioner can be beneficial. Thus, we foresee to benefit from mixed-precision in preconditioners, e.g. packed fma32 is necessary for AMD. We intend to propagate the concept of mixed-precision computations to the underlying Krylov-type solvers as well.

Furthermore, we foresee to employ alternative solvers in the Neko code, e.g. for better scalability or more robust numerics. For instance, p-MultiGrid (p-MG) plus Algebraic MultiGrid (AMG) as the pressure solver.

[3] `https://github.com/ExtremeFLOW/neko`

[4] `http://exaflow-project.eu`

2.4 waLBerla

2.4.1 Description

waLBerla (widely applicable Lattice Boltzmann from Erlangen)[3] is a modern open-source multi-physics simulation software framework with a focus on CFD applications [19]. The main unique feature of waLBerla is its uncompromising focus on large-scale simulations and scalability. It supports the massive parallelism of current peta- and future exascale supercomputers with a framework of carefully designed distributed data structures that allow the implementation of many advanced algorithms. Automated testing ensures the correctness of the functionality across a wide range of target hardware and software environments and makes it well-suited for robust further developments. Its current version also features adaptive techniques. For that it employs a block-structured partitioning of the simulation domain including support for dynamic grid refinement. This also includes functionality for load balancing, for check-point-restart, and even automatic resilience techniques that may become essential on future extreme-scale systems. waLBerla contains efficient, hardware specific compute kernels to achieve optimal performance on most common supercomputing CPU and GPU architectures; the framework[5] is written in C++17 and CUDA. waLBerla has, e.g., been used successfully to implement phase-field and free surface models, and to study the flow around wind turbines. Furthermore, accurate predictions of the flow through porous media like Diesel filters or sediment beds have been obtained and used to improve existing macroscopic models. Simulations of particulate flows have always been a special focus of the group's effort throughout the continuous development of waLBerla. The integration of the rigid body physics engines PE and MESA-PD allows us to model multi-physics scenarios with a granular phase. With this integration, a close and efficient coupling of fluid and particle simulations is possible. This is a key component to realize massively parallel simulations of large-scale particulate systems and constitutes one of the unique features of our framework.

2.4.2 Requirements

Underlying algorithmic solvers and (simplified) test cases There is no dedicated proxy-application for waLBerla. However, it is possible to generate a simple proxy-application doing, e.g., a channel flow or a single particle in a fluid flow simulation. There is Python interface for testing.

To study the underlying algorithmic solutions, we can commence by working on a simplified version of the code with a few small test cases. Such small test cases cover the relevant functionalities of the code for the CEEC project.

Concerning the underlying sovlers in waLBerla, we use explicit time stepping and do not have to solve any systems of equations in the form of $Ax = b$. Thus, there is no need of any preconditioning techniques. The Lattice Boltzmann method is working on regular cells (or an Octree), the particle simulation works on data structures for neighbor lists.

Fault tolerance waLBerla supports check-pointing, i.e. the user is able to specify how often (e.g. after a certain number of time steps) or to configure which data is stored for each check-point. There is also support of fault-tolerant with MPI [11].

[5] `https://i10git.cs.fau.de/walberla/walberla`

2.4.3 Potential strategies

Here, we outline potential strategies for enhancing scalability, numerical properties, energy efficiency, as well as execution time. In order to utilize in full the modern heterogeneous clusters, we foresee to add support for particles on GPUs in the frame of the CEEC project.

We explore a possibility of adapting/cropping the working precision of the waLBerla code. Since the scheme is explicit, the rounding errors have a tendency to accumulate and have a larger impact, especially on the later stages of computations. A feasible scenario is to lower precision at the start of computation and gradually increase it toward its end.

Regarding topology optimization, the adjoint solver (back propagation) would be very interesting for waLBerla. For a note, there are works on the adjoint Lattice Boltzmann method.

We think that waLBerla can benefit from the optimized I/O or compression techniques combined with check-pointing. We intend to explore this at CEEC.

2.5 Nek5000/ NekRS

2.5.1 Description

Nek5000 and NekRS[6] are highly-efficient and scalable open source incompressible and low Mach flow solvers [13] employing the spectral element method (SEM), a high-order weighted residual technique for spatial discretization that can accurately represent complex geometries. Globally, the SEM is based on decomposition of the domain into E smaller subdomains (elements), which are assumed to be curvilinear hexahedra (bricks) that conform to the domain boundaries. Locally, functions within each element are expanded as Nth order polynomials cast in tensor-product form, which allow differential operators on N^3 grid points per element to be evaluated with only $O(N^4)$ work and $O(N^3)$ storage. The principal advantage of the SEM is that convergence is exponential in N, yielding minimal numerical dispersion and dissipation. Significantly fewer grid points per wavelength are required in order to accurately propagate a signal (or turbulent structure) over extended times in high Reynolds number simulations. This advantage was demonstrated in a study at NREL, which showed that, for a given accuracy, turbulent channel simulations performed with 7th-order spectral elements require half as many grid points in each direction as do comparable finite-volume-based simulations.

The solution procedure for solving the governing equations is based on a high-order splitting scheme, where the hydrodynamic equations are advanced with a backward difference/ characteristic-based (BDF/ CHAR), time-stepping algorithm developed for the ALE method [16]. The BDF/ CHAR scheme allows the simulation to overcome CFL restrictions imposed by standard schemes such as backward difference/ extrapolation (BDF/ EXT). Nek5000 is equipped with multilevel solvers that scale to millions of cores. The multilevel solvers require global coarse-grid solves that are based on fast direct solvers developed in Tufo and Fischer [17]. The pressure substep requires a Poisson solve at each step, which is effected through multigrid-preconditioned GMRES iteration coupled with temporal projection to find an optimal initial guess. Particularly important components of Nek5000 are its scalable coarse-grid solvers that are central to parallel multigrid. Counts of 15 GMRES iterations per timestep for billion-gridpoint problems are typical with the

[6] `https://nek5000.mcs.anl.gov/`

current pressure solver. The GPU-oriented NekRS, which is written in C++/OCCA, is the refactored version of Nek5000, which is written in F77/C.

Recent activity involves enhancement of Nek5000's support for combustion and reactive flows, multiphase (liquid/ gas and fluid-particle-particle interactions), multimodel physics (e.g., drift-diffusion, combustion, RANS), and moving domains (e.g., rotating machinery, internal combustion engine, fluid-structure interaction in reactors). In the past several years Nek5000 has been developed further to include Nonconforming Overlapping Schwarz Discretizations, including MultiRate timestepping in Overlapping grids, Advanced Meshing and Mesh Optimization, and a high-order characteristic-based ALE approach for Moving-Domains.

2.5.2 Requirements

Underlying algorithmic solvers and (simplified) test cases For Nek5000, there is an archived proxy-application called nekbone, under the Nek5000 Github organization and could be used for initial testing. There is no proxy app for NekRS but it is possible to derive a simplified scaled down problems.

There are two CEED benchmarks, BP5 (aka nekbone) and BPS5. The NekRS example that can be used for such tests is called Kershaw example. These examples are described in the CEED reports[7][8]. BPS5 is a solver benchmark which is about how fast the Poisson equation can be solved on a deformed mesh. In a sense, it is a proxy of the Nek pressure solver. BP5 is a benchmark problem which uses a simple Jacobi preconditioner. More sophisticated preconditioners are used in BPS5, like GMRES with Schwarz and AMG.

To study the underlying algorithmic solutions and to experiment with mixed-precision and precision cropping strategies, BP5 and BPS5 (Kershaw) benchmarks can be used and be tuned to the runtime within several minutes.

Concerning the underlying solvers in the codes, there is a set of Krylov-type solvers used: PCG for BP5 (only the simple Jacobi preconditioner), GMRES, Schwarz and AMG for BPS5. Few preconditioners are employed along these solvers for better numerical stability and fast convergence. For instance, Jacobi for velocities and scalars; Schwarz with spectral and algebraic MG for pressure. Both the solvers and the preconditioners are tightly coupled within the codes. Furthermore, the solvers are matrix-free; there is no straightforward possibility to form a matrix for testing outside the code.

Fault tolerance Currently, Nek5000/NekRS have no fault tolerance, but rely on regular check-pointing. For instance, field files for restart are typically written every hour but not the entire memory of the application. In case of a hardware (HW) failure, when things go silently wrong, there is no mechanism in the codes to detect such failures.

To elaborate more on the present state of the check-pointing mechanism, check-pointing only contains restart information, not storing the whole system state (no particularly good reason has been identified to do this). There is also a possibility to store the solution at more than one time steps (up to three steps) for cases where a seamless restart is important, e.g. in flow stability studies.

[7] `https://ceed.exascaleproject.org/docs/ceed-ms37-report.pdf`
[8] `https://www.osti.gov/servlets/purl/1845639`

2.5.3 Potential strategies

We outline potential strategies for enhancing scalability, numerical properties, energy efficiency, as well as execution time. In terms of scalability, we consider to work on addressing some of the bottlenecks described earlier, e.g. improve strong scaling gather/scatter type kernels; coarse polynomial and AMG multigrid levels; data layout for low polynomial degrees.

For faster time-to-solution and/or better energy-to-solution, we intend to engage the concept of mixed-precision computations into the underlying Krylov-type solvers and in preconditioners. Particularly, since lowering precision in preconditioners yields to be successful for some cases, we foresee to utilize mixed-precision in preconditioners.

Concerning support of fault-tolerance mechanisms, we would like to explore how to check types of HW failures. For instance, how to check if simulation deadlocks and the solvers stop converging, etc.

3 Strategies definition

Here, we define strategies and present algorithmic approaches toward reliable, resilient, efficient, and fast algorithmic solvers in the consortium CFD codes for Exascale. We cover all the tasks of WP3 and address below algorithmic solvers, mixed-precision, fault tolerance, adaptivity and error control, as well as topology optimization. Timeline for the planned work and corresponding deliverables follows the one from the Grant Agreement.

3.1 Numerical methods and solvers for Exascale

To enable the next generation of exascale CFD simulations, we need to significantly improve existing methods in terms of both scalability and numerical properties. We focus on the scalability issues associated with iterative Krylov methods and the algorithmic challenges of formulating scalable algebraic multigrid methods (AMG) with optimal convergence properties. Specifically, we will address the scalability and numerical issues by considering communication- avoiding and communication-hiding algorithmic variants, including pipelined Krylov formulations. The pipelined methods will be reinforsed with the residual replacement and restarting techniques. In addition, we will develop a novel non-heuristic parallelization of optimal coarsening algorithms for AMG with suitable mixed (combining geometric and algebraic) approaches to enable its use in matrix-free formulations, either as a standalone solver or as a scalable preconditioner inside an iterative Krylov method.

3.2 Mixed-precision algorithms

Due to the energy consumption constraint for large-scale computing that encourages the revision of the architecture design, scientists also review the applications and the underlying algorithms organization. The main aim is to make computing sustainable and apply the lagom principle ("not too much, not too little, the right amount"), especially when it comes to the compute/ storage precision. Thus, we introduce an approach to address the issue of sustainable, but still reliable, computations from the perspective of computer arithmetic tools such as Verificarlo [6] and Verrou [8]. We propose to inspect the consortium codes with the help of computer arithmetic tools to investigate precision appetites

as well as to identify numerical abnormalities. Based on this analysis, we can propose a strategy for cropping precisions. One promising strategy is to reduce the data movement but still compute in full precision. For implicit iterative solvers, reducing precision of data storage for a matrix of preconditioners is one of promising choices. For explicit solvers, we consider to start with lower precision and increase the working precision during the course of computations in order to mitigate the effect of round-off accumulation. These efforts will lead to the revision of algorithms. Such strategy can be reinforced with communication-avoiding and/or -hiding strategies such as pipelined methods, e.g. in the Neko code. This way the communication and, hence, data movement will be decreased even further. To accommodate the requirement of more accurate computation, e.g. 10^{-8} and higher, the pipelined methods can be reinforsed with the residual replacement and restarting technique. Alternatively, we can leverage the iterative refinement procedure and floating-point expansions [1].

$$\boxed{\text{Inspection with tools}} \rightarrow \boxed{\text{Strategy}} \rightarrow \boxed{\text{Revision of algorithms}}$$

Overall, our strategy for mixed-precision and, thus, energy-efficient and/ or faster computations can be summarized in the following four steps.

1. Apply **arithmetic tool to the code** $\rightarrow$ manual/ automatic in order to inspection precision appetites.
2. If the reduction is possible, **derive and apply algorithmic mixed-precision solutions** following the suggestions from the tools. Here we can also explore a possibility of using **iterative refinement** to tune the accuracy to the desired level while using lower precisions.
3. Conduct **probabilistic (aka optimistic) error analysis**. This is a fascinating but a bit out of scope topic to the CEEC project:
   - The error bound with constant $\sqrt{n}\mu$ with high probability instead of $n\mu$ as in case of summation.
4. Implement on **heterogeneous hardware**, possibly with stochastic rounding that randomly maps x to one of two bounds.

The main focus of the algorithmic development within CEEC is on the steps 1, 2, and 4 that will result in algorithmic solutions and implementations for scalable, faster, and possibly more energy-efficient computations in the consortium codes.

Compared to the algorithmic works on mixing precisions and still deriving the same accuracy as with the original (often double) precision, we will explore a possibility of sacrificing few bits of accuracy for faster and more energy efficient executions under the constraint of numerically reliable and meaningful computations.

3.3 Fault-resilience algorithms

With respect to fault-tolerant or fault-resilient algorithms, most CEEC codes rely on a basic strategies like check-pointing together with the generation of restart files at particular instances. Only the FLEXI code relies on the evaluation of a performance index with the option to terminate simulation in case this performance index exceeds the given threshold. In addition, our codes offers the possibility to restart with a different number of MPI tasks. This approach is a very helpful feature in a post-fault scenario where a simulation has to be restarted on a different number of nodes as initially intended. The

check-point and restart approach does not include a pre-fault or fault detection step and includes the restart of the entire simulation. In total, none of the codes offers any advanced fault tolerance approaches to react or recover instantaneously on a faulty node for example via a local mitigation strategy. Thus, advanced resilience and fault tolerance strategies, as summarized for example in [4], are required and have to be incorporated into the CEEC codes. To develop or include advanced fault tolerance or fault-resilient algorithms the following step-wise approach is proposed:

1. Analysis of classical fault mechanisms on HPC systems of interest with the help of hardware producers and involved computer centers. Within this step, a list of fault mechanisms on the different HPC systems of interest is generated with the aim to obtain an overview on possible faults to be expected.
2. Classification of fault mechanism. The list of fault mechanisms is reviewed in terms of relevance, like the mean time between failure (MTBF) for example, and the potential to recover from the CEEC applications, i.e., software point of view. The resulting rating will serve as an input for mitigation strategies to be applied.
3. Analysis of needed software-based, i.e., user-level mitigation strategies, frameworks and tools. Basing on the rating from the previous step suitable mitigation strategies are identified and proposed. This strategies might be available already via existing frameworks and tools like Fenix or plain MPI functionalities for example. On the other hand further developments might be needed. Here, the CEEC-code PyCOMPSs (BSC) may be considered to re-distribute tasks from a faulty node to spare nodes. Beside these node-level strategies one might also think about algorithmic resilience.
4. Development and integration of selected approaches and testing. Once mitigation strategies are identified these strategies are married with the CEEC codes. Afterwards testing of implemented functionalities basing on fault-injection frameworks like FAIL is proposed.
5. Application within the LHC simulations. The integrated mitigation strategies will applied within the simulation of the lighthouse cases.

3.4 Adaptivity and error control

Large-scale nonlinear simulations of industrially relevant turbulent flows at realistic Reynolds numbers are computationally extremely expensive and currently intractable. They have to be performed with highly parallel codes based on accurate discretization methods of high order and require special techniques allowing for minimization of the computational effort and optimal usage of the resources. Although automated mesh adaptivity offers the potential for improved accuracy at reduced cost and its benefits have been known for three decades, it is not widely used due to software complexity, stability issues and lack of proper error estimators and efficient refinement strategies for complex geometries. CEEC will focus on enabling robust adaptive at exascale, guided by accurate error indicators. Developing scalable adaptive mechanisms specific to high-order methods by combining adaptation in polynomial order (p-adaption) and element topology (h-adaption), enabling efficient tracking of complex flow features at a low computational cost. In addition, these mechanics will be combined with novel solution-driven adaptation in floating-point precision (fp-adaption), further reducing the computational and energy costs. The new algorithmic development will build upon previous work concentrating on goal-oriented es-

timation techniques e.g., based on the adjoint and spectral error estimators for high-order methods [15, 14], and will mainly be integrated in Neko to reduce the computational cost of LHC6.

3.5 Scalable optimization algorithms

The main algorithmic development needed for the successful completion of LHC3 — topology optimization of static mixers — will be based on resolving some optimization-specific issues and providing components to be integrated into the Neko code, as outlined below.

1. The first task is to provide Neko with an *adjoint-equation solver* for the efficient computation of sensitivities, preferably using an exact formulation not assuming pointwise divergence-free velocity fields as in Nek5000.
2. Also, an efficient *check-pointing scheme* is paramount. The same strategy as in Nek5000, based on the Revolve algorithm [9], will be used also here.
3. Additional attention must be given to issues concerning the *unstable nature of the adjoint equation* for chaotic transient problems. Strategies include simplification, in which the chaotic motion is suppressed for the adjoint evaluation, or the use of time-averaged sampling.
4. The optimization of the fluid mixer requires the solution to an additional *scalar transient thermal transport equation*. The problem is one-way coupled and hence, the thermal transport does not couple back into the forward fluid problem. However, its contribution must be accounted for in the adjoint sensitivity analysis.
5. Another equally important issue concerns the high-order nature of the spectral finite element method. In order to achieve a sufficiently fine design representation, *multi-level* and/or *multi-resolution parameterizations* will be investigated. This work includes strategies to suppress under- and overshooting of design variable bounds and the development of efficient convolution type filters in a massively parallel setting.
6. An efficient *optimization algorithm* must be integrated in order to achieve the optimized design within a minimum amount of time and with a minimum use of computational resources. The algorithm could involve the utilization of a multi-level design representation, such that slow moving structural features on the fine grid can advect faster using coarse grid information.

4 Summary

In this deliverable, we have provided analysis of the CEEC codes and underlying solvers with the engagement and outputs from the code owners in the form of the detailed questionnaire. Based on the provided answers and regular internal discussions, we have derived both requirements and potential strategies for improving algorithms and for efficient exploitation of Exascale architectures for each individual code. Furthermore, we have defined approaches toward reliable, resilient, efficient, and fast algorithmic solvers in the consortium CFD codes for Exascale, covering algorithmic solvers, mixed precision, fault tolerance, adaptivity and error control, as well as topology optimization.

The next steps in WP3 are focused on the implementation of the defined strategies and approaches, integration into the CEEC codes, and their validation and verification within

the lighthouse cases at scale.

Bibliography

[1] Roman Iakymchuk, María Barreda, Stef Graillat, José I. Aliaga, and Enrique S. Quintana-Ortí. Reproducibility of Parallel Preconditioned Conjugate Gradient in Hybrid Programming Environments. *IJHPCA*, 34(5):502–518, 2020.

[2] Hartwig Anzt, Jack Dongarra, Goran Flegar, Nicholas J. Higham, and Enrique S. Quintana-Orti. Adaptive precision in block-jacobi preconditioning for iterative sparse linear system solvers. *Concurrency and Computation: Practice and Experience*, 31(6):e4460, 2019.

[3] Martin Bauer, Sebastian Eibl, Christian Godenschwager, Nils Kohl, Michael Kuron, Christoph Rettinger, Florian Schornbaum, Christoph Schwarzmeier, Dominik Thönnes, Harald Köstler, et al. walberla: A block-structured high-performance framework for multiphysics simulations. *Computers & Mathematics with Applications*, 81:478–501, 2021.

[4] Tommaso Benacchio, Luca Bonaventura, Mirco Altenbernd, Chris D Cantwell, Peter D Düben, Mike Gillard, Luc Giraud, Dominik Göddeke, Erwan Raffin, Keita Teranishi, et al. Resilience and fault tolerance in high-performance computing for numerical weather and climate prediction. *The International Journal of High Performance Computing Applications*, 35(4):285–311, 2021.

[5] M Blind, P Kopper, D Kempf, M Kurz, A Schwarz, A Beck, and CD Munz. Performance improvements for large scale simulations using the discontinuous galerkin framework flexi. *High Performance Computing in Science and Engineering*, 22, 2022.

[6] Yohan Chatelain, Eric Petit, Pablo de Oliveira Castro, Ghislain Lartigue, and David Defour. Automatic exploration of reduced floating-point representations in iterative methods. In Ramin Yahyapour, editor, *Euro-Par 2019: Parallel Processing - 25th International Conference on Parallel and Distributed Computing, Göttingen, Germany, August 26-30, 2019, Proceedings*, volume 11725 of *Lecture Notes in Computer Science*, pages 481–494. Springer, 2019.

[7] CEEC Consortium. Deliverable D1.1 – CEEC exascale lighthouse cases and their needs, August 2023. Available via `https://ceec-coe.eu/`.

[8] François Févotte and Bruno Lathuilière. Debugging and optimization of HPC programs with the verrou tool. In Ignacio Laguna and Cindy Rubio-González, editors, *2019 IEEE/ACM 3rd International Workshop on Software Correctness for HPC Applications (Correctness), Denver, CO, USA, November 18, 2019*, pages 1–10. IEEE, 2019.

[9] A. Griewank and A. Walther. Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Software*, 26(1):19–45, 2000.

[10] Niclas Jansson, Martin Karp, Artur Podobas, Stefano Markidis, and Philipp Schlatter. Neko: A modern, portable, and scalable framework for high-fidelity computational fluid dynamics. *ArXiv*, 2107.01243, 2021.

[11] Nils Kohl, Johannes Hötzer, Florian Schornbaum, Martin Bauer, Christian Goden-

schwager, Harald Köstler, Britta Nestler, and Ulrich Rüde. A scalable and extensible checkpointing scheme for massively parallel simulations. *The International Journal of High Performance Computing Applications*, 33(4):571–589, 2019.

[12] Nico Krais, Andrea Beck, Thomas Bolemann, Hannes Frank, David Flad, Gregor Gassner, Florian Hindenlang, Malte Hoffmann, Thomas Kuhn, Matthias Sonntag, and Claus-Dieter Munz. Flexi: A high order discontinuous galerkin framework for hyperbolic–parabolic conservation laws. *Computers & Mathematics with Applications*, 81:186–219, 2021.

[13] James W. Lottes, Paul F. Fischer, and Stefan G. Kerkemeier. Nek5000 webpage, 2008. `http://nek5000.mcs.anl.gov`.

[14] Nicolas Offermans, Daniele Massaro, Adam Peplinski, and Philipp Schlatter. Error-driven adaptive mesh refinement for unsteady turbulent flows in spectral-element simulations. *Computers & Fluids*, 251:105736, 2023.

[15] Nicolas Offermans, Aadam Peplinski, Oana Marin, and Philipp Schlatter. Adaptive mesh refinement for steady flows in nek5000. *Computers & Fluids*, 197:104352, 2020.

[16] Saumil Patel, Paul Fischer, Misun Min, and Ananias Tomboulides. A characteristic-based spectral element method for moving-domain problems. *Journal of Scientific Computing*, 79(1), 12 2018.

[17] H.M Tufo and P.F Fischer. Fast parallel direct solvers for coarse grid problems. *Journal of Parallel and Distributed Computing*, 61(2):151–177, 2001.

[18] Mariano Vázquez, Guillaume Houzeaux, Seid Koric, Antoni Artigues, Jazmin Aguado-Sierra, Ruth Arís, Daniel Mira, Hadrien Calmet, Fernando Cucchietti, Herbert Owen, Ahmed Taha, Evan Dering Burness, José María Cela, and Mateo Valero. Alya: Multiphysics engineering simulation toward exascale. *Journal of Computational Science*, 14:15–27, 2016. The Route to Exascale: Novel Mathematical Methods, Scalable Algorithms and Computational Science Skills.

[19] Markus Wittmann, Viktor Haag, Thomas Zeiser, Harald Köstler, and Gerhard Wellein. Lattice boltzmann benchmark kernels as a testbed for performance analysis. *CoRR*, 2017.