DLR-PROJECT DIGITAL-X

NEXT GENERATION CFD SOLVER 'FLUCS'

T. Leicht[1], D. Vollmer[1], J. Jägersküpper[1], A. Schwöppe[1], R. Hartmann[1], J. Fiedler[2], T. Schlauch[3]
DLR (German Aerospace Center)
[1]Institute of Aerodynamics and Flow Technology
Lilienthalplatz 7, 38108 Braunschweig, Germany
[2]Institute of Propulsion Technology
Linder Höhe, 51147 Köln, Germany
[3]Simulation and Software Technology
Linder Höhe, 51147 Köln, Germany

Abstract

The development of DLR's 'next-generation' flow solver was initiated as part of the project Digital-X [1] to provide a basis for a consolidated flow solver using modern software techniques with high flexibility and high degree of innovation for a wide range of multidisciplinary applications. An overview of the design and development of the resulting flow solver Flucs (FLexible Unstructured CFD Software) is presented, its current status is described, and first results for internal and external flows are shown.
The development followed a top-down approach identifying significant drivers in terms of application range and software design and was evaluated during the project to identify possible drawbacks in early stages and is continuously monitored to keep maintainability and expandability. The development is supported by modern software tools, such as distributed version control, web-based code reviews, and continuous integration. The kernel of the resulting design is a framework whose data structures and methods serve as a basis for implementing lean modules, for example equations, discretizations and time-integration methods. The framework provides basic functionalities like efficient implementation of loops, parallelization, or the provision of required data. Based on the framework, two discretizations are implemented: a second-order finite-volume discretization and a discontinuous Galerkin discretization with variable order, both of them using the same sets of implemented equations like the Euler-equations, the Navier-Stokes equations, or the RANS equations. A focus of the next-generation solver is its efficient use on current and future parallel HPC systems. The framework currently provides a two-level parallelization consisting of a domain decomposition that features communication/computation overlap, and shared-memory parallel processing of a domain. The simulation-setup layer of Flucs is designed as compatible Python API for the simulation environment FlowSimulator [2] which provides a flexible interface to a wide range of multidisciplinary simulation-scenarios.

ACKNOWLEDGMENTS

The development of the next generation CFD solver Flucs would not have been possible without contributions from many colleagues. The authors would like to thank all who supported the discussions and development activities over time with various levels of involvement in the development itself and the software development environment, namely T. Gerhold[1], E. Hoffmann[1], P. Kelleners[1], F. Spiering[1], A. Stück[1], N. Kroll[1], T. Kaller[2], F. di Mare[2], M. Meinel[3], M. Klitz[3], M. Litz[3].

1. INTRODUCTION

The multidisciplinary project Digital-X (04/2012-06/2016) [1] represents a basic component for the progressive realization of the vision of digital aircraft design and virtual flight testing at DLR. The project focused on the development and industrialization of advanced simulation methods and processes to pioneer their application for exploring the whole flight envelope virtually. Part of its activities was addressing the first development steps of DLR's next generation CFD (Computational Fluid Dynamics) solver Flucs (FLexible Unstructured CFD Software) to provide a basis for a consolidated flow solver with high flexibility and high degree of innovation for a wide range of multidisciplinary applications using modern software techniques and utilizing upcoming HPC architectures.

In parts of Digital-X, CFD was taken as given. Compared to other disciplines it has been used for a long time and is relatively mature but still has room for improvement, in particular regarding current and upcoming progress in computer hardware, numerical algorithms and software development. The unstructured DLR CFD solver TAU [3], which is routinely used in industry and research, was further improved within Digital-X. However, integrating recent developments for HPC (High Performance Computing) hardware, implicit solver techniques or higher-order discretizations into legacy codes with regard to a sustainable software development can be a difficult task. Also considering expanded requirements of multidisciplinary scenarios this motivated an innovative strategy for developing a next generation CFD solver.

Higher-order methods have been investigated world-wide over recent years [4]. In particular, the Discontinuous Galerkin method has potential for reliable error estimation and adaptivity [5], as well as scale-resolving simulations [6]. DLR has its own experience due to collaboration in several European projects like ADIGMA [7], IDIHOM [8], or TILDA and participation and hosting of international workshops on higher-order CFD methods [4], [9], [10], [11], [12]. Additionally, DLR has gained experience regarding the implementation of higher-order CFD codes in several prototype codes [13], [14]. What was missing up to now was the close integration into existing technologies like process chains for multidisciplinary simulations.

The potential of implicit schemes is well known and their usage within CFD solvers on modern hardware has been shown lately [14], [15], [16]. Their ability to converge to a solution at all and to reduce the dependency on solver settings for convergence as well as on meshes and their quality can improve reliability and robustness. The modified usage of multigrid schemes as algebraically motivated formulation [14], [16], or the inclusion of linear multigrid also for non-linear problems have the ability to further improve reliability and robustness of the numerical methods [14], [17]. Such numerical algorithms touch the basic formulation and assumptions of a code and it can be difficult to retrofit them in a large-scale legacy code afterwards.

The architectures of modern HPC systems are changing towards realizing multiple levels of parallelization. In addition to the distributed-memory level which has been used for decades since the advent of cluster systems, today in particular the shared-memory level at which multiple CPU cores of a multi- or many-core CPU make use of the same memory has gained attention. Domain decomposition using message passing (MPI), which originally addressed distributed-memory parallelization, has been the standard approach for decades. It has long been known, however, that addressing the shared-memory level differently within a hybrid parallelization strategy can be beneficial in terms of parallel performance, e.g. [19]. For an improved parallel implementation, alternative communication and memory-access patterns have been considered for modern high-performance simulations. Experiences from the projects HiCFD [20] and GASPI [21] and corresponding prototypes show the potential of a multi-level parallelization. On the other hand, it has also become apparent that it is difficult, if not almost impossible, to realize such benefits enabled by modern HPC systems within an existing large-scale legacy code that is based on an MPI-only parallelization.

Advanced CFD software has to cover a wide range of multidisciplinary scenarios these days. Modern software engineering provides techniques to support a modular code development towards highly flexible and maintainable software [28]. Large legacy codes developed over a long period of time can be difficult to maintain e.g. due to possibly unclear interactions of features or code parts, chronologically and logically unrelated development activities, which could have profited from closer collaboration, or the lacking involvement of an integrator with substantial overall code knowledge for new development activities. Additionally, it can be difficult to apply modern forms of quality assurance [40] to a legacy code after most of it has already been written. Although it is common practice to use a testing environment and a code-review process, their subsequent application within development of legacy codes can be difficult to realize.

These aspects of modern CFD-software development together with prior experiences in many aspects of CFD software development [2], [3], [13], [14], [27], [36], [37], [38] motivated a concerted effort in Digital-X into the new common environment Flucs with particular focus on exploiting current and upcoming HPC architectures, attention to interactions of different disciplines, integration of code for internal and external flows, and on an innovative design to plan for the future and for future extensibility. A focus is on quality assurance, testing and maintainability to develop Flucs as an investment into a sustainable capability for CFD. Among these innovative aspects the transfer of established methods and models of the TAU code into Flucs is considered.

The development and current status of Flucs is presented in the following order: First an overview of the development process realization is given. Then, the main design decisions and their results are described. The status of the implemented spatial discretizations and iterative solvers is presented together with several simulation results. Finally, concluding remarks are offered.

2. PROCESS REALIZATION

A development process realization consists of individual parts from defining and managing requirements to specifying the design, selecting tools which support the actual code implementation, establishing a testing environment, organizing communication among developers, defining a documentation environment and various evaluation steps like code reviews, performance checks and overall software evaluations. In particular, Flucs followed a top-down approach consisting of identifying design drivers in terms of application range and software design, specifying requirements and functionality, implementing and evaluating a prototype and reworking the prototype based on the evaluation. The development process itself is supported by an infrastructure made up of various tools to automate the process as much as possible.

2.1. Requirements Management

Requirements for Flucs were gathered on the basis of the experience of different groups. Design drivers were identified by the development team and formulated as project plan. The development team conducted about 20 dedicated interviews with experienced, in-house and industrial CFD practitioners. Additionally, developers from existing codes like TAU were interviewed. The results were condensed into a requirement specification with more than 150 individual requirements. Long-term goals such as complex simulation scenarios for a ‘helicopter in maneuver’ were assessed to complete the requirement specification. The prioritization of the requirements yielded a functional specification from which 104 issues were classified as high-priority requirements. These issues were entered into the web-based issue-management tool Mantis [26] and organized within the tool to roadmaps defining the content of different prototypes and releases. The resulting long-term plan is well beyond the scope of Digital-X, but with detailed planning for the duration of the project. During Digital-X the focus was on tackling design drivers and not on maximizing functionality.

2.2. Version control and code reviews

Both version control and code reviews are basic components for current software development [28], [29]. For Flucs the distributed version control of source code is done via the tool git [31] which supports different version-control work-flows. The central code version is maintained with a clean history and coherent steps of moderate size, while local backups, branches and merges can be exploited to the taste of individual developers.

The review process follows a four+ eyes principle [29]. Each commit into the central Flucs development branch is checked by at least one additional developer. The process itself can be seen as an iterative development cycle, which consists of a code-change proposal, its review and a corresponding improvement, typically with more than two or three iterations. A review considers both form and content and is supported via the Gerrit [33] web-interface. The web-interface provides visualizations of differences between proposed code and the base version as well as differences between subsequent iterations. It allows inline comments to discuss individual code parts. Furthermore, automated testing is integrated via the continuous integration tool Jenkins [25]. Each commit has to pass the testing environment before a review begins. One-click download of the proposed code version for local testing by the reviewer is also provided. Several further features are available to facilitate the review process and make it as little time-consuming as possible.

2.3. Testing

In general, a suite of automated tests is required for quality assurance and as a basis for code extensions and design changes, in particular in a growing team of developers. Following [28], the testing environment of Flucs defines different categories of testing: build testing, functional testing, and formal testing.

Build testing uses different compilers to check basically for errors and warnings and additionally, compatibility of the code against the standard of the programming language. Furthermore, the exchange of individual network-communication libraries like MPI vs. GASPI is provided. Functional testing includes unit tests, integration tests and system tests. Unit tests use the Google Test library [22] for expected behavior or expected failure modes of individual methods and classes. Integration tests check the interplay of different methods and classes. System tests run the full code as intended and check output against independent references. Formal testing checks the code for typical programming mistakes and the coding standard using static code analysis tools Cppcheck [22] and Vera++ [24].

The complete testing environment is fully integrated into the web-interface for code-review via Jenkins [25] and runs for each commit proposed for the central code version and designated points in time, e.g. every night.

2.4. Core team, communication and documentation

The development of Flucs is handled by a small core team of experienced developers with overall code knowledge. At least one core-team developer acts as reviewer for each commit to avoid problems based on too little knowledge of existing code or corresponding concepts. Complex extensions have to be done in collaboration between the developer and the core team. Extensions in foreseen areas, e.g. adding a new turbulence model or system of equations, are more easily performed by a developer and can directly jump into the review process.

Meetings in form of a phone conference of the core team with currently involved developers are conducted weekly. In these meetings the status of each active development is discussed and further development steps are arranged. Additionally, there are frequent meetings of smaller groups of developers involved in a particular development task (collaborative development). During Digital-X a breakout-review-session after mid-term was conducted. The aim was to identify weak points in the current design and the development process itself, in particular after the start-up phase.

All weekly meetings are documented in a dedicated Wiki. Code reviews are documented in Gerrit, requirements and roadmaps are documented via Mantis that is also used as bug-tracking system. The documentation generator Doxygen [34] is used to write the API API (Application Programming Interface) documentation of Flucs itself directly within the code.

3. DESIGN

The design of Flucs includes abstractions and their interactions derived from the identified design drivers and the definition of the requirement specification. The identified main design drivers were multi-disciplinary simulations, a modular software design with high level of abstraction to facilitate testing, maintenance and extensibility, multi-level HPC support, an efficient overset-grids technique, mixed-element unstructured grids with hanging nodes, higher-order discretizations via a Discontinuous Galerkin method, and implicit solution algorithms based on consistent derivatives via AD (Automatic Differentiation). Harmonized with design solutions to the diverse requirements these led to fundamental decisions for Flucs overall, but in particular concerning consistency aspects, HPC support, the abstraction of discretizations and derivative information.

3.1. Fundamental decisions

Future CFD applications are predominantly found inside a multidisciplinary setting. DLR's integration platform for MDA (Multi-Disciplinary Analysis) is the FSDM (Flow Simulator Data Manager) [2]. Flucs is designed as FSDM-plugin to provide best possible compatibility with this type of workflow. To avoid inconsistencies from the beginning no stand-alone version exits. Instead the simulation set-up layer is implemented as control layer (Fig. 1) in Python [30]. Accordingly, the data exchange between FSDM and Flucs is organized via the FSMesh-object (Fig. 2).

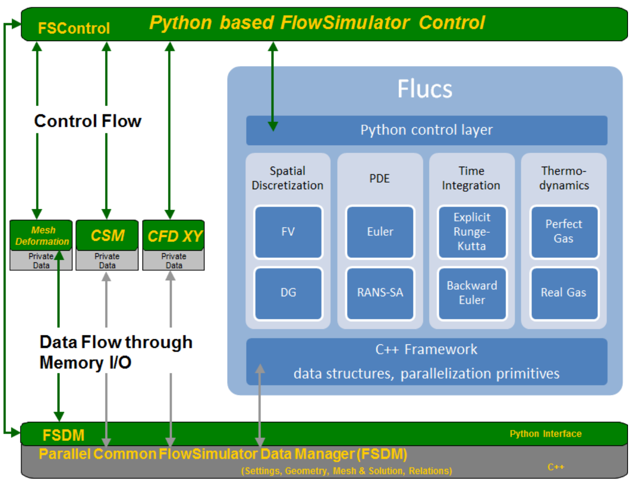

Fig. 1 Flucs as FSDM-plugin

While Flucs acts as a single component within the FlowSimulator environment, it has itself a similar modular structure consisting of a Python control-layer, a common framework and modules with exchangeable implementations. The common framework defines and implements interfaces to handle data structures and the HPC layer. The exchangeable implementations of the actual solver modules organize the underlying governing equations, additional algebraic equations and combinations of those, and the spatial and temporal discretizations, cf. (Fig. 1).

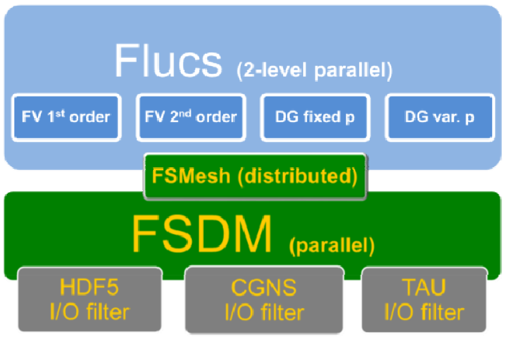

Fig. 2 I/O interface of the FSDM-plugin Flucs

The spatial discretization module provides a Finite Volume and a Discontinuous Galerkin discretization and has to handle mixed-element unstructured grids according to the corresponding design drivers. For the Finite Volume discretization the cell-centered grid metric was chosen, partly due to the fact that a closer relation with the Discontinuous Galerkin discretization can be exploited. Extensions have to be done in FSDM along with the Flucs development in order to make consequent use of the data flow of FSDM. In particular, an improved parallel output, a modified mesh partitioning for the cell-centered grid metric, the support for high-order curvilinear mesh elements and for hanging nodes, as well as polynomial data for visualization is necessary.

In order to enable the high level of abstraction and still allow the generation of efficient machine code, C++11 [30] is chosen as underlying programming language. Templates are heavily used to minimize run-time overhead of many abstractions.

3.2. Consistency

In the past, several problems within existing codes could be traced back, sometimes after long investigation, to some simplification or small scale inconsistency in a code that did not show an adverse effect at the time of integration. Further issues were related to the usage of an algorithm or a data structure optimized for a specific task but also used for similar tasks and to the usage of dependent but outdated variables simultaneously. To avoid such code inconsistencies as effectively as possible Flucs refrains from pre-mature optimizations and simplifications. Furthermore, Flucs supports tracking of potential write access to field vectors holding flow data to avoid stale copies of remote data. Centrally managed network communication of the distributed data is done when required. In addition, primitive variables, which include redundant information to conservative variables, are set up from conservatives for a given local state once and cannot be changed independently. While gradients are directly available in Discontinuous Galerkin discretizations, a Finite Volume discretization typically has to precompute them. In order to avoid outdated gradient information, Flucs supports tracking of potential write access to field vectors and updates precomputed gradients upon the next request.

3.3. HPC support

An important motivation for developing Flucs is the efficient use of current and upcoming parallel HPC systems. CFD applications are globally coupled which requires frequent communication and synchronization. Hence, it is important but difficult to achieve high efficiency on massively parallel systems. Modern HPC architectures feature multiple levels of parallelism: a node-to-node level, a multi-core level, and a SIMD (Single Instruction Multiple Data) level, cf. (Fig. 3). For an efficient code all levels of parallelism need to be exploited. Different techniques are required to exploit the full potential of the different levels. Concepts exist for the treatment of all three levels [20], [21].

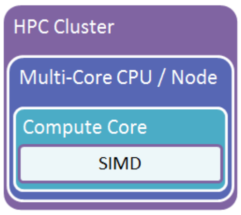

Fig. 3 Levels of parallelism

The node-to-node level is based on a domain decomposition of the grid with halo layers. The communication between the nodes is usually realized via a network communication library. Flucs defines a common abstraction of different network communication libraries to be able to replace a specific library easily. The focus is on the usage of the GASPI standard in form of its reference implementation GPI [35] to exploit asynchronous communication with overlap of communication and computation. The standard MPI [18] is a fallback option. The multi-core level is based on a subdomain decomposition of a node-level grid domain. The memory is shared between the cores of a node, so that no data duplication via halo overlap is needed. Instead one-sided write operations at subdomain boundaries are implemented within Flucs to avoid data races. The SIMD level to exploit vectorization within a single compute core is currently realized only partially, by using Eigen [41] as a dense linear algebra package with SIMD support.

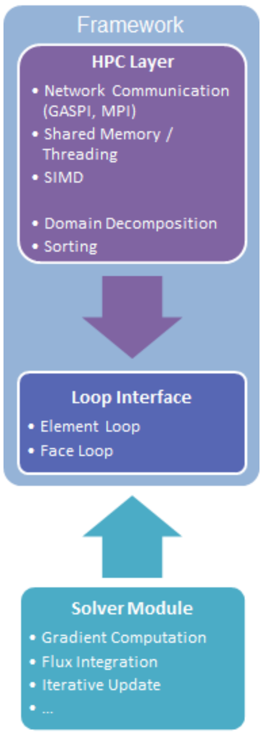

Fig. 4 Separated HPC layer

A key-element of Flucs' HPC support is to hide details from the developers outside the core team and the solver code itself. Many CFD operations follow simple patterns: based on local input compute local output and repeat this over the whole mesh; usually in the form of loops over all elements or faces. Parallel programming requires additional synchronization instructions before and/or after a loop- and possibly reorganization of underlying data. Consequently, the amount of logic around the actual operation grows for each level of parallelization. To keep the logic separated from the solver code and hidden from developers, an HPC layer is introduced, cf. (Fig. 4). The HPC layer separates what is done locally (the loop body, local input to local output) from how it is done globally. The how-logic is implemented just once in the HPC layer of the code framework (loop interface) to make it possible that only the loop interface has to be extended or exchanged for porting to new architectures.

3.4. Abstract discretization

Flucs provides a first or second order Finite Volume discretization and a Discontinuous Galerkin discretization of order one or higher. To exploit as many similarities between these discretizations and to be as consistent as possible (Sec. 3.2) an abstract design was introduced, cf. (Fig. 5).

The basis of each discretization is the computational mesh holding data of integration points in elements and integration points on faces. Each face connects either two elements or one element to a boundary. In the integration points the state variables and their gradients are needed. State variables are provided for each integration point via a field vector holding conservative variables. Fluxes over faces have to be evaluated and integrated so that at each face left and right variables (and their gradients) in each integration point are needed. How the variables and gradients at a given point in the mesh are evaluated from the field vector depends on an ansatz chosen for an element plus (potentially) additional reconstruction. For a Finite Volume discretization this can be a cell average in combination with a limited linear reconstruction, while a polynomial ansatz is used for a Discontinuous Galerkin discretization. Reconstructed variables and gradients are passed to closures to compute augmented variables like pressure. The augmented set of variables is used by a PDE (Partial Differential Equation) object to evaluate fluxes and source terms. For evaluating convection fluxes, the directivity of convection is taken into account depending on the underlying PDE and closures via an eigen-value decomposition of the corresponding flux. The actual discretization only combines a mesh, an ansatz selecting reconstruction, a PDE and corresponding closures, and an upwinding scheme if the PDE has a convection term. Depending on the discretization the reconstruction includes special treatments of variables, like adding a face gradient correction or a lifting operator, or applying a slope limiter etc.

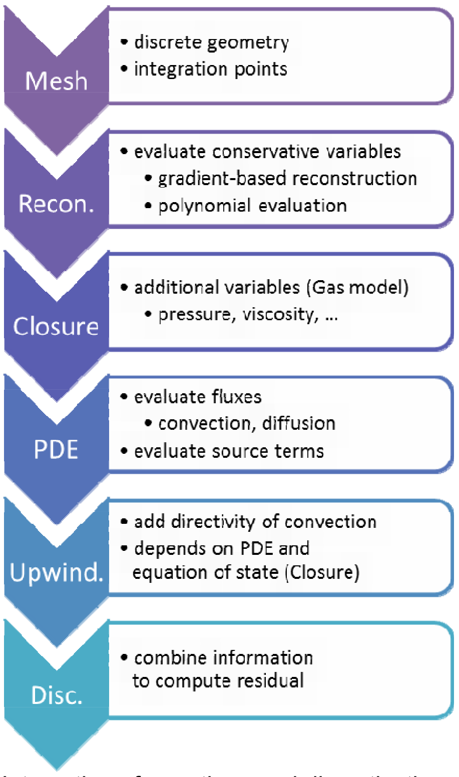
Mesh
• discrete geometry
• integration points
Recon.
• evaluate conservative variables
• gradient-based reconstruction
• polynomial evaluation
Closure
• additional variables (Gas model)
• pressure, viscosity, ...
PDE
• evaluate fluxes
• convection, diffusion
• evaluate source terms
Upwind.
• add directivity of convection
• depends on PDE and equation of state (Closure)
Disc.
• combine information to compute residual

Fig. 5 Interaction of equations and discretizations

The treatment of boundaries is also abstracted via a reconstruction. Just the matching (right) exterior state at a boundary face has to be provided dependent on the interior (left) state and the specific type of boundary condition. The same flux as on interior faces is then computed to incorporate boundary conditions. This principle holds also for overset grids. In that case the right state at a face is obtained from reconstruction in another mesh (block). This works with any non-negative overlap (including zero overlap) and is a great simplification for the meshing procedure. Also local (output) quantities at the wall as well as integral quantities are evaluated based on the same flux computation for inner and boundary faces [38].

3.5. Automatic differentiation

Within a CFD solver, consistent derivative information is required for various reasons. The full derivative is needed for adjoint problems to support gradient-based optimization and for reliable error estimation and mesh adaptation. A compact-stencil derivative-approximation for the Jacobian is typically utilized by implicit solvers [15]. The assembly of the Jacobian based on derivative information is a two-step procedure. First, differentiate the local residual contribution. Second, sort the local contribution into the global Jacobian. The second step can be implemented once for a given discretization. Derivative information needed for the first step can be provided via differentiation by hand or via AD. Differentiation of the code by hand is simple in theory, but cumbersome and error-prone in practice. Flucs realizes the first step by AD in forward mode via operator overloading. This yields exact derivatives up to machine precision. Currently Eigen::AutoDiff [41] is used, but can be replaced by other implementations.

4. STATUS AND RESULTS

Finally, an overview of Flucs' status concerning implemented discretizations, iterative solvers and the HPC support is given and simulation results for designated test cases are presented.

The abstract discretization of Flucs (Sec. 3.4) includes a Roe-upwinding for convection and a central discretization for diffusion. Several governing equations for compressible flow in a fixed frame of reference are implemented: Euler (inviscid flow), Navier-Stokes (laminar viscous flow), and Reynolds-averaged Navier-Stokes plus Spalart-Allmaras turbulence model (turbulent viscous flow). The perfect gas equation of state, the viscosity computation via Sutherland's law and the thermal conductivity computation via Prandtl number are implemented as algebraic closures. The following boundary conditions are implemented: fixed state (which is used to model far-field boundaries), total pressure and temperature based inflow with pressure based outflow (Riemann), free slip wall, adiabatic no-slip wall, symmetry plane including exact discrete equivalence with full model, periodic boundary pairs (for pure translation) and an overset-grids artificial boundary.

4.1. Finite Volume discretization

The Finite Volume discretization is specialized from the abstract discretization (3.4) as second-order discretization using a gradient-based, piecewise linear reconstruction. Gradients are computed based on the Green-Gauss theorem. The typical slope limiters like the limiter of Barth and Jespersen [42] and Venkatakrishnan [43] for strong shocks can be applied. The central discretization for diffusion corrects the averaged face gradient directionally [44].

In order to show that results based on Flucs' Finite Volume discretization are plausible, a cruise-flight test case with a turbulent flow around the NASA Common Research Model (CRM) was computed. Results for Flucs' overset-grids boundary treatment for matching boundaries and overlapping boundaries are shown for an internal flow around a linear cascade of T106A turbine blades and for a wing configuration with deflected flaps.

4.1.1. Cruise flight

A turbulent flow around the NASA CRM, a wing-body configuration, with Mach number 0.85, an angle of attack of 2.209°, and Reynolds number 5.0*10^6, taken from the Fifth AIAA CFD Drag-Prediction Workshop (DPW5) [45], is considered. The angle of attack is motivated by results for the target lift of 0.5 [15]. The grids are from the DPW5 web site [45].

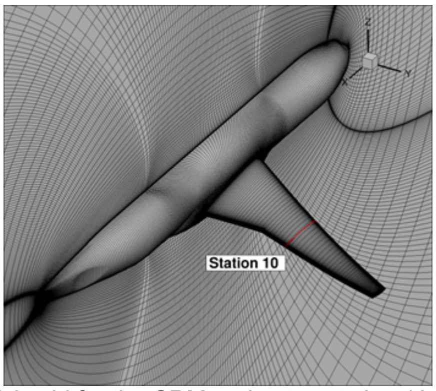

Fig. 6 L3 grid for the CRM and cut at station 10

Fig. 6 shows a plot of the L3 grid (medium grid level) including the cut at station 10 on the wing for which pressure-coefficient distributions of Flucs results are plotted in Fig. 7 and Fig. 8.

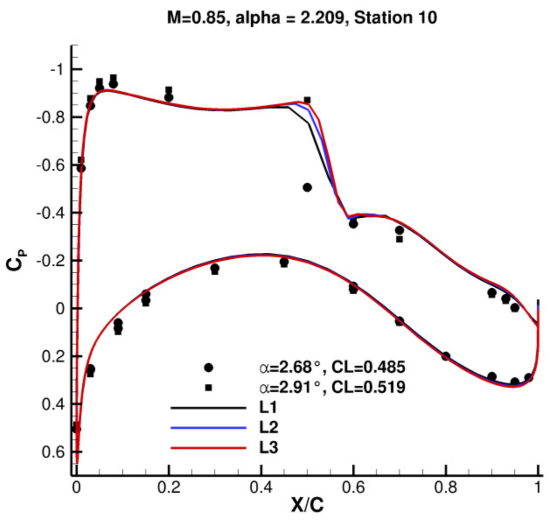

Fig. 7 Pressure-coefficient distributions at station 10

Flucs results are presented for the grid levels L1-L3 (tiny to medium grid level) in Fig. 7. As reference, also wind tunnel results for two runs close to the target lift of 0.5 are included, though the conditions of measurement and computation are actually rather different [45]. The results are reasonable compared to the wind tunnel data and show an expected convergence behavior from grid level L1 to grid level L3 with respect to a steeper resolution of the shock on the finer grid levels.

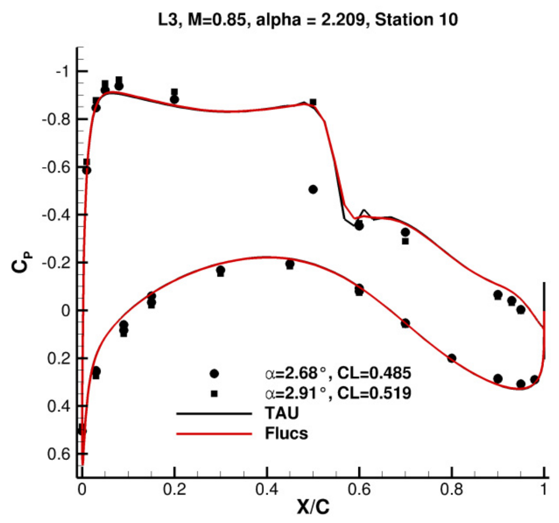

Fig. 8 Pressure-coefficient distributions of Flucs, TAU and wind tunnel results

For comparison to an established flow solver, TAU and Flucs results are plotted for the medium grid level L3 in Fig. 8. Except for the expected differences at the shock due to different discretizations of discontinuities, where Flucs uses a Roe-upwinding versus TAU's central scheme with matrix dissipation [15], and at the trailing edge due to different grid metrics, the results are hardly distinguishable.

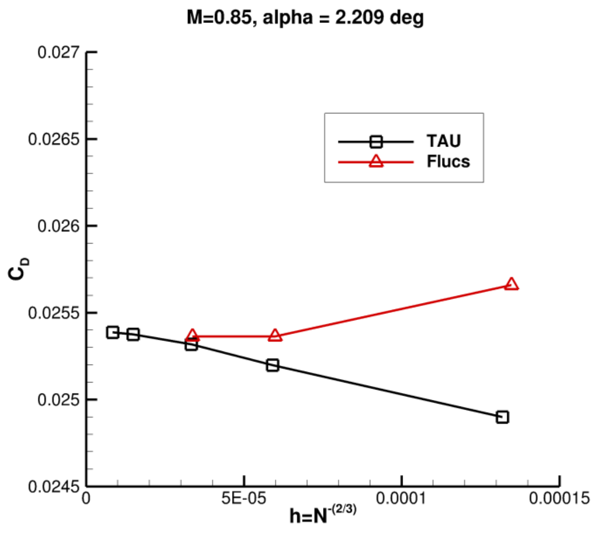

Fig. 9 Grid convergence of drag coefficient

The grid convergence of the drag and lift coefficients computed with Flucs on grid levels L1 to L3 are shown in Fig. 9 and Fig. 10 and compared to TAU results computed on grid levels L1 to L5. Though results on the finest grid levels are not yet computed with Flucs, both lift and drag coefficient tend to converge to similar values as TAU does, with similar levels of error for a given grid.

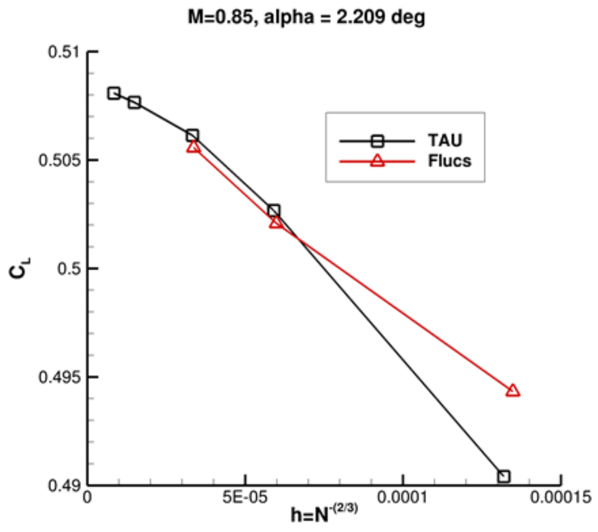

Fig. 10 Grid convergence of lift coefficient

Altogether, the comparison of Flucs to wind tunnel and TAU results for the cruise-flight test case shows that Flucs results are reasonable.

4.1.2. Linear cascade

An internal flow around a linear cascade of T106A turbine blades, a 2D approximation of a cylindrical cut through a stator, is considered.

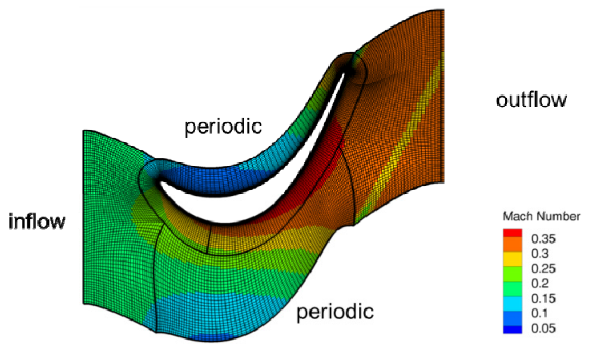

Fig. 11 Structured multi-block grid and Mach number distribution

The multi-block grid consisting of five blocks with matching interfaces and 18060 hexahedrons is shown in Fig. 11. The five blocks are treated as separate meshes and coupled via Flucs' overset-grids boundary treatment. The coupling is flux conservative in this case due to the matching block interfaces. In order to map the periodic boundaries for this linear cascade, again the overset-grids boundary treatment is used, now with a fixed translation (offset) vector.

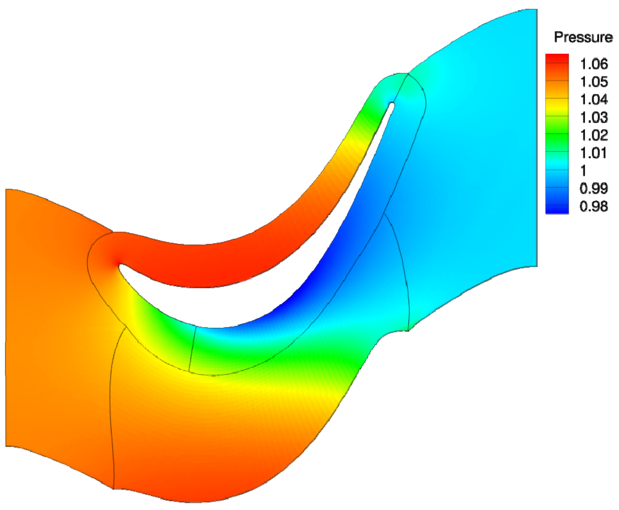

Fig. 12 Pressure distribution

In- and outflow are treated with Flucs' Riemann boundary treatment. At a Mach number of 0.3 (outflow), the parameters are chosen such that at the outflow the non-dimensional pressure and temperature are 1. The third dimension of the 2D case is treated via symmetry planes. The Reynolds number is set to 100,000.

The pressure distribution of the resulting flow field is plotted in Fig. 12 and the resulting distribution of the pressure coefficient over the blade's surface in Fig. 13,

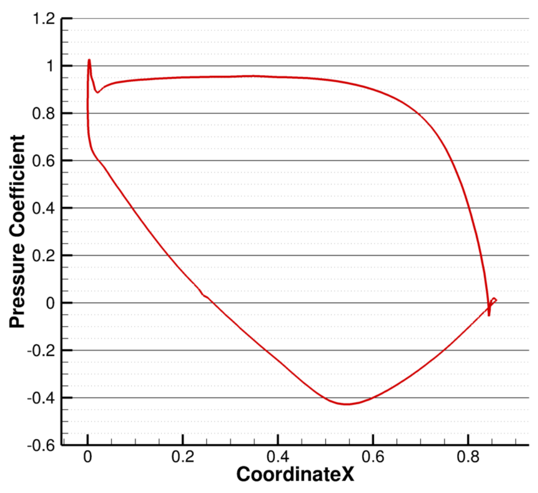

Fig. 13 Pressure coefficient distribution on the blade

From a qualitative perspective, the simulation shows a reasonable drop in pressure with corresponding increase in Mach number which is typical for such a turbine. Even the characteristic pressure drop at the round trailing edge is captured as can be seen in Fig. 14. As no results in the literature using the same turbulence model (negative Spalart-Allmaras, which does not allow to prescribe a turbulence intensity of the inflow) were available, however, a quantitative comparison with results from literature is not included here.

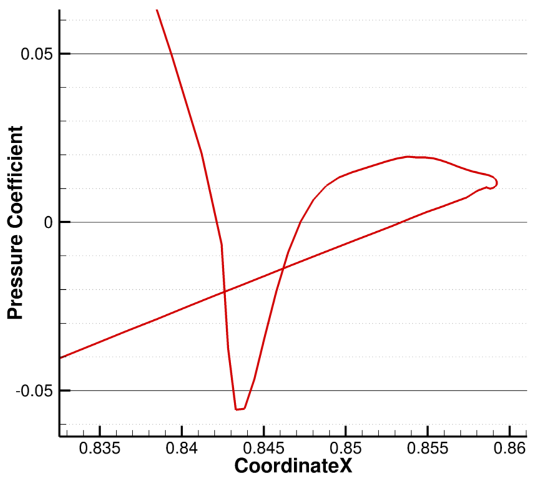

Fig. 14 Pressure coefficient distribution at the trailing edge

4.1.3. Overset grids with overlap

The demonstrated handling of multi-block meshes with zero overlap via the overset-grids technique should be regarded as a corner case of the corresponding implementation. In the following, a wing with two flaps is considered, with gaps between wing and flaps as well as between the two flaps. The set-up of the three grids with predefined holes (allowing for different flap-deflection angles due to the overlapping regions) is plotted in Fig. 15.

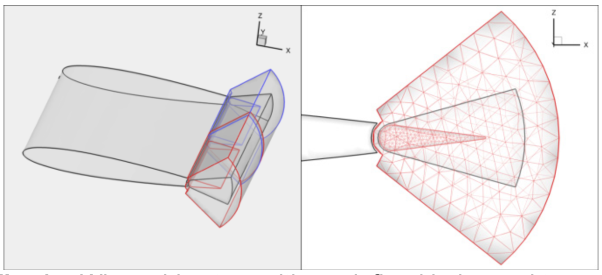

Fig. 15 Wing grid set-up with predefined holes and two component meshes

The pressure distribution on the lower surface and streamlines through the gap are shown in Fig. 16 with both flaps at a position of 10°.

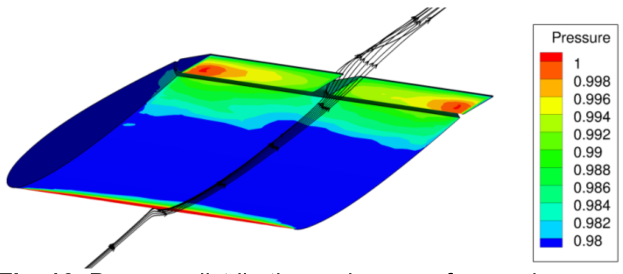

Fig. 16 Pressure distribution on lower surface and streamlines through gap

Fig. 17 shows the corresponding pressure distribution on the upper surface. In contrast to the coupling of matching mesh interfaces (Sec. 4.1.2), in this setting Flucs' overset-grids technique is not flux preserving.

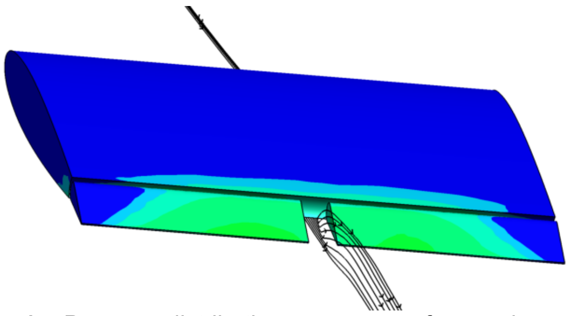

Fig. 17 Pressure distribution on upper surface and streamlines through gap

4.2. Discontinuous Galerkin discretization

The Discontinuous Galerkin discretization is specialized from the abstract discretization (3.4) using a modal basis for the polynomial reconstruction. This basis is defined in physical space (not on a reference element) and thus suitable for agglomerated grids. Compile-time fixed polynomial degree design orders generated from a generic template are available for design order 1 (for Euler), 2, 3, and 4. The central discretization with direct differentiation will be extended to BR1 [46] and BR2 [47]. Currently, only straight-sided grids can be handled. The metric computation during the preprocessing requires extensions for curvilinear meshes whereas the mesh data structure and discretization are already fully prepared.

In order to show that results based on Flucs' Discontinuous Galerkin discretization for different design orders are plausible a turbulent flow around the L1T2 high-lift airfoil is computed. The potential of the higher-order Discontinuous Galerkin discretization for scale-resolving simulations is shown via the computation of a Taylor-Green vortex.

4.2.1. L1T2 high-lift airfoil

A 2D turbulent flow around the L1T2 three-element high-lift airfoil is considered. Results for the Discontinuous Galerkin discretization in Flucs are compared to results of the Finite Volume discretization and experimental data to show that they are reasonable for different orders and to show a potential gain of accuracy per degrees of freedom if higher-order discretizations are used.

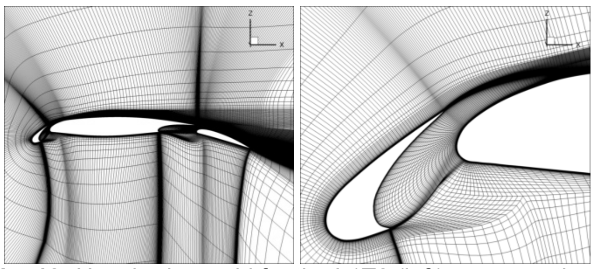

Fig. 18 Hexahedron grid for the L1T2 (left), zoom zo the slat (right)

A sequence of three grids is used, a coarse level of 4268 hexahedrons, a medium level of 17072 hexahedrons and a fine level of 68288 hexahedrons. The grid around the entire airfoil and around the slat is plotted in Fig. 18.

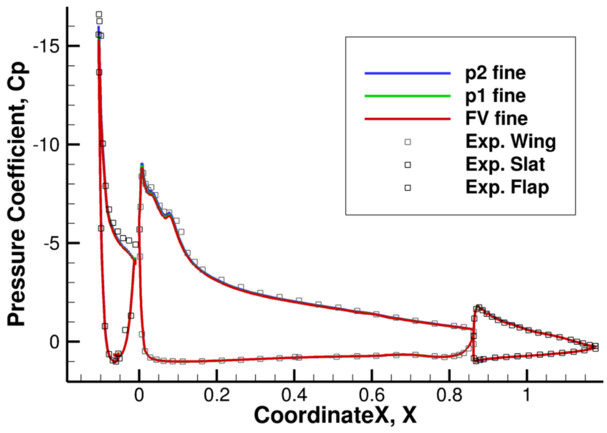

Fig. 19 Comparison of pressure-coefficient distributions of DG with design order of 2 and 3, FV, and experimental results on the fine grid

The flow is computed for Mach number 0.197, an angle of attack of 20.18° and Reynolds number $3.52*10^6$. The pressure-coefficient contribution on all wing components is plotted in Fig. 19 for the second order Finite Volume (FV) discretization and the second and third order Discontinuous Galerkin (DG) discretization on the fine grid. Additionally, data from a wind tunnel experiment is plotted. The simulation results and experimental data match reasonably well. Differences between the resulting pressure-coefficient distributions for the three discretizations are small. This is somewhat expected, since the fine grid was originally generated for a Finite Volume computation of this case.

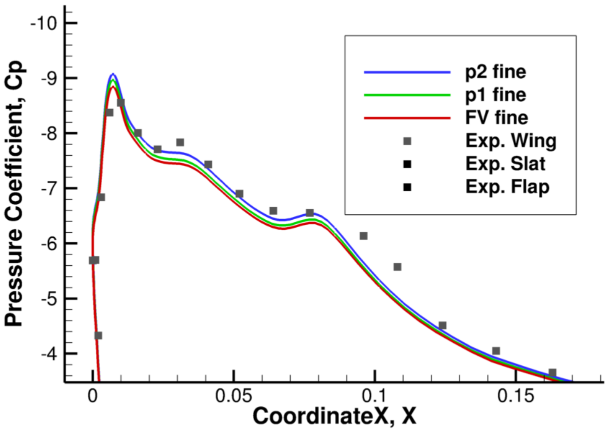

Fig. 20 Comparison of pressure-coefficient distributions of DG with design order of 2 and 3, FV, and experimental results at the leading edge of the main wing

Fig. 20 illustrates the differences in greater detail. A certain improvement towards the experimental data can be seen for increasing resolution with higher-order Discontinuous Galerkin discretizations.

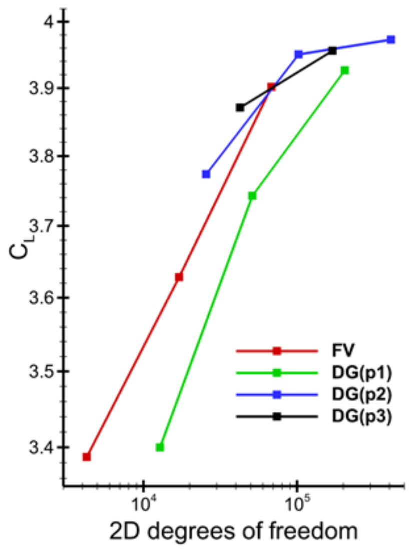

Fig. 21 Grid convergence of lift-coefficient

The grid convergence of the lift coefficient for various spatial discretization schemes is shown in Fig. 21 against the number of degrees of freedom per equation. With increasing number of degrees of freedom all discretizations tend to converge to a reasonable lift coefficient. Whereas the second-order results for both the Discontinuous Galerkin and the Finite Volume discretization are close together regarding the obtained lift coefficient on a given grid, the higher-order Discontinuous Galerkin results show, as expected, a clear gain of accuracy regarding the ratio of lift coefficient and number of degrees of freedom on the same grids. This is despite the fact that only straight-sided grids are used. Further improvement is expected with curvilinear grids.

4.2.2. Taylor-Green vortex

The Taylor-Green vortex is often used to assess turbulence scale resolving capabilities of CFD codes [4]. It models the decay of large coherent vortex structures (given as an analytical initialization) into smaller structures and the transition to turbulence. Here, essentially incompressible conditions at Mach number 0.1 are assumed with a Reynolds number of 1,600. Flucs is used in ILES (Implicit Large Eddy Simulation) mode, i.e. no subgrid-scale model is employed. Results obtained for various discretizations are then compared to reference results obtained from a resolution-converged DNS (Direct Numerical Simulation) of this case by a spectral code.

The 4th order (in space) Discontinuous Galerkin simulation (via cubic polynomial ansatz functions, “DG p3 5M” in Fig. 22) is run on a 63^3 Cartesian grid (treated as an unstructured one, of course). It uses 20 degrees of freedom (DoFs) per element (per equation), resulting in a total of about 5M DoFs (per equation). To demonstrate the superior resolution of the Discontinuous Galerkin method, a 2nd order Finite Volumes simulation is run on a 171^3 grid (“FV 5M”), which also results in about 5M DoFs. Moreover, the coarser 63^3 grid is used with this Finite Volume method (“FV 5M/20”, merely about 250K DoFs). All simulations use the classic 4th order explicit Runge-Kutta time integration. The evolution of the field’s enstrophy over time is a widely accepted error measure for this test case [4]. As can be seen from Fig. 22, 5M DoFs are obviously not sufficient to fully resolve all structures. Yet this is not the point here as both DG and FV use Roe's scheme for a fair comparison, and not a more apt low-dissipation scheme. The simulations clearly confirm the expectation that, in this ILES, the specific Discontinuous Galerkin method makes considerably “more use” of DoFs than the specific FV method does.

While it is trivial to create meshes for this simple case at any required resolution, a single existing unstructured mesh might be all that is available for a given complex case in practice. Here, the potential of the Discontinuous Galerkin scheme to obtain a tremendous improvement of the effective resolution on the same given mesh is relevant. In that sense, the specific Discontinuous Galerkin discretization can make “more use” of a mesh that the specific Finite Volume method, albeit at substantially increased cost.

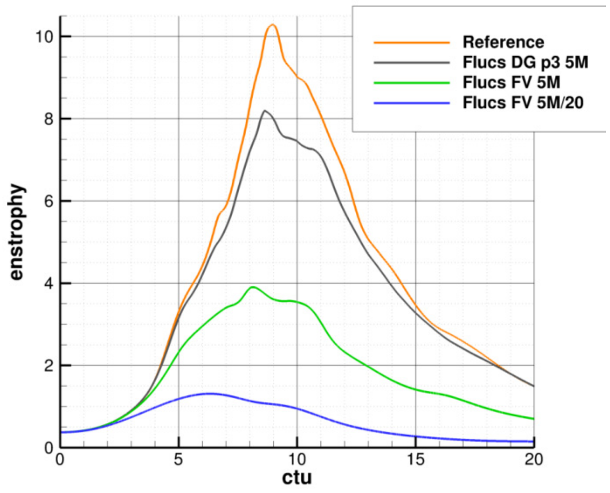

Fig. 22 Enstrophy over convective time units (ctu) during ILES of Taylor-Green vortex decay

4.3. Iterative solver

Currently two different iterative solvers are available in Flucs: an explicit multi-stage Runge-Kutta method and an implicit linearized backward Euler method. For steady cases local pseudo-time steps are computed based on a stability estimate and a CFL number. For unsteady cases a uniform time step is employed.

The explicit multi-stage Runge-Kutta method is implemented in a generic way, i.e. based on different number of stages and Runge-Kutta coefficients various different explicit iterative solvers are available. The implicit linearized backward Euler method is based on an approximate Jacobian corresponding to a compact stencil obtained via AD. The CFL number is computed adaptively based on the residual reduction using SER (Switched Evolution Relaxation) [48]. The linear system is solved with a simple damped element-block-Jacobi iterative solver. Further development of the implicit iterative solver is planned in the follow-on project VicToria.

Two test cases are chosen to verify that the implemented methods show expected behavior.

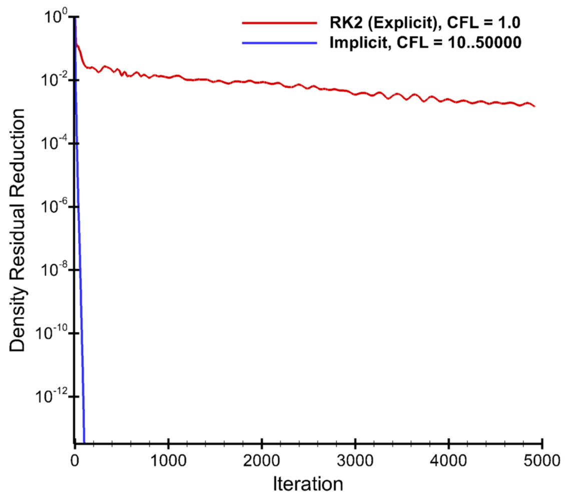

Fig. 23 Convergence history of the density residual against iteration number (Finite Volume discretization), inviscid flow around wing

The 3D inviscid flow around a wing resolved by an unstructured grid is considered. Two computations with Flucs' Finite Volume discretization are compared. The first uses an explicit 2-stage Runge-Kutta method with CFL number 1. The second uses the implicit method with a CFL number up to 50000. The comparison of the density residual reduction against the number of iterations cf. (Fig. 23) and against the computation time cf. (Fig. 24) shows expected behavior. The implicit method yields a tremendous reduction in the number of iterations required to converge the residual. This gain in iteration steps translates into reduced but still significant savings in computation time compared to the explicit method.

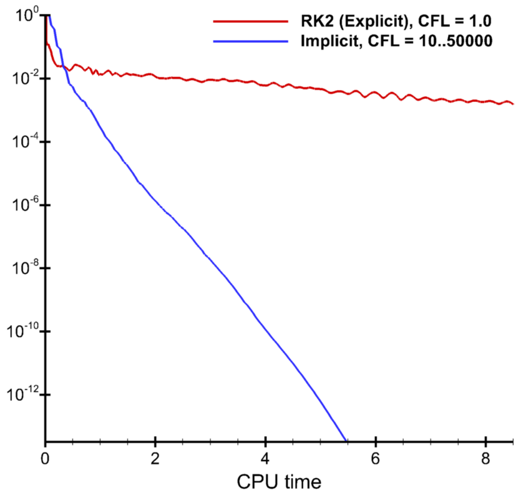

Fig. 24 Convergence history of the density residual against computation time (Finite Volume discretization)

A 2D turbulent flow around the L1T2 three-element high-lift airfoil resolved by a block-structured grid is considered. Flucs' Discontinuous Galerkin discretization is used with design order 3 on the finest grid. Due to the compact stencil of the Discontinuous Galerkin method, the exact derivative of the residual is used as Jacobian within the implicit method with a CFL number increasing to 1000. The residual norms against the number of iterations of the five conservative variables are shown in Fig. 25. The convergence behavior is as expected but the limitations of the current implementation, mainly the linear solver, are visible. Previous experience shows that this case can be converged in the order of 100 iterations with an extended implicit method including multigrid techniques [8].

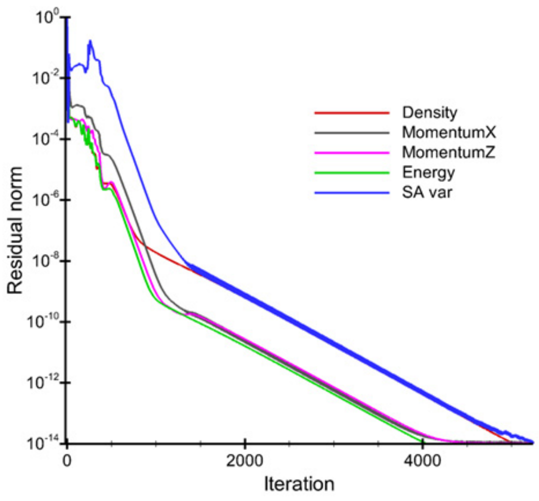

Fig. 25 Convergence history of residual norms against number of iterations (Discontinuous Galerkin discretization), turbulent flow around L1T2 airfoil

Both test cases show that the basic methodology works as expected, in particular with respect to AD. Extensions of the implicit method are planned within the follow-on project VicToria.

4.4. Parallel scalability

Parallel performance has been one of the major design drivers of Flucs. In addition to the known concept of domain decomposition to make use of distributed-memory machines like HPC clusters, Flucs features a second level of domain sub-decomposition which splits each domain into subdomains. All subdomains of a single domain must reside in the same memory, however, since Flucs makes use of so called "threads" to compute a domain in parallel. All threads dealing with the subdomains of a domain are run on the same multicore CPU, usually such that there is a one-to-one mapping of software thread to "processing element" of this CPU. The benefit of this second level is that there are fewer domains, namely just one per multicore CPU, which results in less communication over the network and reduced process-synchronization overhead. To opt out performance, load imbalances on this second level (subdomains) as well as thread-synchronization overhead must be as small as possible, of course – just as for the domain-decomposition level.

The efficiency of the new second-level shared-memory parallelization is shown in Fig. 26. Flucs' 2nd order Finite Volume discretization (of the Reynolds-averaged Navier-Stokes plus Spalart-Allmaras equations) is run using explicit time integration for the L1 CRM mesh, cf. Sec. 4.1.1. A single domain is used (i.e. no domain decomposition) to utilize a single IBM "Power A2" chip (found in IBM's massively parallel BG/Q architecture). This CPU has 16 compute cores, each of which can simultaneously run four threads, resulting in a total of 64 threads. Thus, here we consider a “strong scaling” scenario as we (try to) use more and more computational resources to solve a fixed-size problem. As the four threads running on the same CPU-core share some of the core’s hardware resources, one may not expect a perfect speed-up of 4. The obtained 70% intra-core parallel efficiency (speed-up 2.82) when running a single domain split into four subdomains on a single CPU-core can in fact be considered a good performance. More interesting is the speed-up obtained when using multiple CPU-cores (each running 4 threads). With an obtained inter-core parallel efficiency of 89% (speed-up 14.24) when using all 16 cores, Flucs shared-memory parallel processing of a domain is able to make almost perfect use of this CPU.

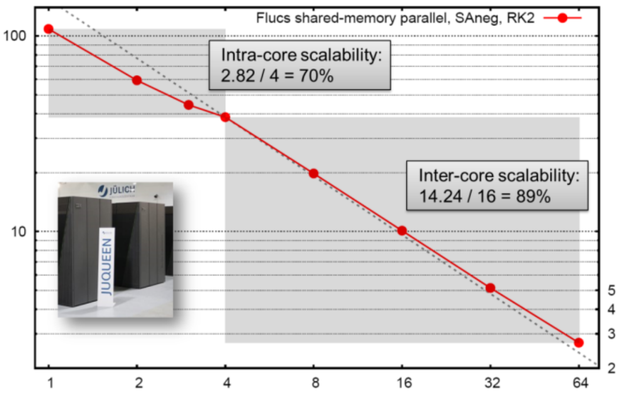

Fig. 26 Runtime over number of threads used; single IBM BG/Q chip: 16 cores, each running 4 threads

The effect of running Flucs in various parallel modes is shown in Fig. 27. Flucs can use two different libraries to exchange data at domain boundaries, namely “MPI” and “GASPI”. Moreover, Flucs allows overlapping the domain-decomposition communication (via one of the two) with computation, which is in fact Flucs’ standard mode. To demonstrate the effect of this innovative feature, simulations were run with no overlap (“NO”) by disabling this overlap. Finally, the simulations were run single-threaded (“ST”), i.e. the second level, namely the shared-memory level, was switched off, resulting in running one domain per CPU-core. Thus, “MPI ST NO” resembles the classical parallelization concept of most legacy codes like, e.g., DLR’s production code TAU.

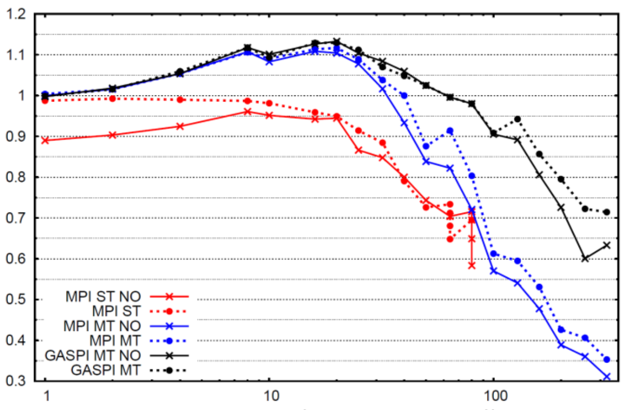

Fig. 27 Parallel scalability of Flucs: parallel efficiency over number of HPC cluster nodes used

Note that one cluster node consists of two 12-core Intel Ivy Bridge EP CPUs, which means that one node has 24 CPU-cores, which run 48 threads due to Intel’s “hyper-threading”. The effect of overlapping communication with computation is always present (solid vs. dashed lines). For the ST runs (red, shared-memory parallelization disabled), it is present for low core counts, whereas for the two-level parallel runs, the effect is seen for high core counts, which is the design goal. Moreover, the effect of the added second parallelization level (black/blue vs. red) is significant. Finally, the effect of using GASPI instead of MPI for domain decomposition is also significant. All in all, running Flucs on a coarse mesh consisting of less than 2 million elements with fully enabled parallelization (black circles) allows using 200 cluster nodes (4800 cores running 9600 threads, i.e. merely 200 elements per thread) with 80% parallel efficiency. With MPI (blue), only 80 nodes can be used at the same efficiency. Without the second level (sub-decomposition), merely 40 cluster nodes can be used at 80% efficiency. These results, which were obtained on the C²A²S²E cluster system [49] located at DLR’s Braunschweig site, demonstrate clearly that the efforts on Flucs’ advanced parallelization concept pay off in terms of a significantly improved parallel efficiency.

5. SUMMARY AND NEXT STEPS

An overview of the design and development of DLR’s next generation flow solver Flucs was presented, details on its current status were given and results for internal and external flows were shown. The reasons for starting the new software Flucs were mentioned. The process realization to support the Flucs development as well as the basic design was presented. The process may seem rather complex involving a multitude of software tools, but process and tool selection were carefully designed to provide a real benefit for the overall development effort. Additionally, the process is not regarded as rigid and can be adapted to better suit the needs, if required. This has already happened in the project duration. Flucs is designed and implemented as a pure FSDM plugin. The plugin is built from a common framework and light-weight solver components to separate details from the ordinary developer and keep the kernel components replaceable if needed. Additionally, a high level of abstraction is used to allow a high level of code reuse. Since the start of Flucs’ development a sustainable design is considered more important than early features. Hence, code consistency has a very high priority and premature simplifications, assumptions and optimizations were not the focus during Digital-X. Finally, the provided examples showed expected results in the new environment.

Having worked out the underlying design aspects, the main focus of further development of Flucs in the follow-on project VicToria will shift towards the extension of functionality. The planned implementations of implicit solver techniques follow the ideas in [15], [16]. The physical models will be extended by a differential Reynolds stress turbulence model [50], [51] and an enhanced transition-modeling transport-equation approach [52], [53]. A focus is on the generalization to a rotating frame of reference and grids under general movement and deformation. Further developments are the fan integration and the calculation of unsteady flows. An important extension will be the support for curved grids for Discontinuous Galerkin discretizations and the implementation of mesh adaptation with hanging nodes for all discretizations. All extensions of functionality are prerequisites to reaching the milestone defined within VicToria of simulating a helicopter in forward flight.

Additionally, during VicToria a dedicated HPC-focused linear algebra package as common building block for various applications will be developed and subsequently used by Flucs.

6. REFERENCES

[1] Kroll, N., Abu-Zurayk, M., Dimitrov, D., Franz, T., Führer, T.,Gerhold, T., Görtz, S., Heinrich, R., Ilic, C., Jepsen, J., Jägersküpper, J., Kruse, M., Krumbein, A., Langer, S., Liu, D., Liepelt, R., Reimer, L., Ritter, M., Schwöppe, A., Scherer, J., Spiering, F., Thormann, R., Togiti, V., Vollmer, D., Wendisch, J.-H.: DLR project Digital-X: towards virtual aircraft design and flight testing based on high-fidelity methods. CEAS Aeronautical Journal, 7, 3-27, 2015

[2] Meinel, M., Einarsson, G.O.: The FlowSimulator framework for massively parallel CFD applications. In: PARA 2010. PARA2010, 6.-9. Juni 2010, Reykjavik, Island, 2010

[3] Schwamborn, D., Gerhold, T., Heinrich, R.: The DLR TAU Code: Recent Applications in Research and Industry. In proceedings of European Conference on Computational Fluid Dynamics, ECCOMAS CDF 2006, Delft, The Netherland, 2006

[4] Wang, Z.J., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Carey, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H.T., Kroll, N., May, G., Persson, P.-O., van Leer, B., Visbal, M.: High-order CFD methods: current status and perspective, International Journal for Numerical Methods in Fluids 72(8), 811-845, 2013

[5] Fidkowski, K., Darmofal, D.: Review of output-based error estimation and mesh adaptation in Computational Fluid Dynamics, AIAA Journal 49(4), 673-694, 2011

[6] Carton de Wiart, C., Hillewaert, K., Bricteux, L., Winckelmans G., Implicit LES of free and wall-bounded turbulent flows based on the discontinuous Galerkin/symmetric interior penalty method, International Journal for Numerical Methods in Fluids 78(6), 335-354, 2015

[7] Kroll, N.: ADIGMA - A European project on the development of adaptive higher-order variational methods for aerospace applications, 47th AIAA Aerospace Sciences Meeting, AIAA 2009-176, 2009

[8] Kroll, N., Leicht, T., Hirsch, C., Bassi, F., Johnston, C., Sørensen, K., Hillewaert, K.: Results and conclusions of the European project IDIHOM on high-order methods for industrial aerodynamic applications, 53rd AIAA Aerospace Sciences Meeting, AIAA 2015-293, 2015

[9] 1st International Workshop on High-Order CFD methods, Nashville, Tennesse, Jan. 7–8, 2012, http://zjwang.com/hiocfd.html, Accessed 08 September 2016

[10] 2nd International Workshop on High-Order CFD methods, Cologne, Germany, May 27–28, 2013, http://www.dlr.de/as/hiocfd/, Accessed 08 September 2016

[11] 3rd International Workshop on High-Order CFD methods, Orlando, Florida, Jan. 3–4, 2015, https://www.grc.nasa.gov/hiocfd/, Accessed 08 September 2016

[12] 4th International Workshop on High-Order CFD methods, Heraklion (Crete), Greece, June 4-5, 2016, https://how4.cenaero.be/, Accessed 08 September 2016

[13] Hartmann, R., Held, J., Leicht, T., Prill, F.: Discontinuous Galerkin methods for computational aerodynamics — 3D adaptive flow simulation with the DLR PADGE code, Aerospace Science and Technology 14 (7), 512-519, 2010

[14] Wallraff, M., Leicht, T.: Higher-order multigrid algorithms for a discontinuous Galerkin RANS solver, 52nd AIAA Aerospace Sciences Meeting. AIAA 2014-936, 2014

[15] Langer, S., Schwöppe, A. Kroll, N.: Investigation and comparison of implicit smoothers applied in agglomeration multigrid in the framework of the DLR TAU-Code, AIAA 2014-0080, 52nd AIAA SciTech, National Harbor, Maryland, January 13-17, 2014

[16] Langer S.: Agglomeration multigrid methods with implicit Runge-Kutta smoothers applied to aerodynamic simulations on unstructured grids, J. Comput. Phys. 277, 72-100, 2014.

[17] Mavriplis, D., Mani K.: Unstructured Mesh Solution Techniques using the NSU3D Solver, AIAA 2014-0081, 52nd AIAA SciTech, National Harbor, Maryland, January 13-17, 2014

[18] Message Passing Interface Forum, http://www.mpi-forum.org/, Accessed 08 September 2016

[19] Jin, H. et al.: High performance computing using MPI and OpenMP on multi-core parallel systems, Journal on Parallel Computing, Vol.37(9), 562–575, Elsevier, 2011

[20] Basermann, A. et al.: HICFD: Highly Efficient Implementation of CFD Codes for HPC Many-Core Architectures. In: Competence in High Performance Computing 2010: Proceedings of an International Conference on Competence in High Performance Computing, 1–13, Springer, 2012

[21] Alrutz, T. et al: GASPI: A Partitioned Global Address Space Programming Interface. In: Facing the Multicore-Challenge III, Lecture Notes in Computer Science, Vol.7686, 135–136, Springer, 2013

[22] Google C++ Testing Framework. https://github.com/google/googletest/, Accessed 08 September 2016

[23] Cppcheck - A tool for static C/C++ code analysis. http://cppcheck.sourceforge.net/, Accessed 08 September 2016

[24] Vera++ - A programmable tool for verification, analysis and transformation of C++ source code. https://bitbucket.org/verateam/vera/wiki/Home/, Accessed 08 September 2016 2016

[25] Jenkins - An automation engine with a plugin ecosystem. https://jenkins.io/, Accessed 08 September 2016

[26] Mantis Bug Tracker. https://www.mantisbt.org/, Accessed 08 September 2016

[27] Jägersküpper, J., Simmendinger, Ch.: A Novel Shared-Memory Thread-Pool Implementation for Hybrid Parallel CFD Solvers, Proceedings Euro-Par 2011, Lecture Notes in Computer Science, Band 6853, Springer, 182-193, 2011

[28] Rajlich V.: Software Engineering: The Current Practice, Chapman & Hall, CRC Innovations in Software Engineering and Software Development Series, ISBN 9781439841228, 2011

[29] Rigby, P.C., Bird, C.: Convergent contemporary software peer review practices, Proceedings of the

2013 9th Joint Meeting on Foundations of Software Engineering, 202-212, New York, 2013

[30] The Python Programming Language, https://www.python.org/, Accessed 08 September 2016

[31] ISO/IEC JTC1/SC22/WG21: ISO/IEC 14882:2011 - The C++ Programming Language, 2011

[32] Git - A distributed version control system, https://git-scm.com/, Accessed 08 September 2016

[33] Gerrit Code Review - Code Reviews for Git, https://www.gerritcodereview.com/, Accessed 08 September 2016

[34] Van Heesch D., Generate documentation from source code, http://www.stack.nl/~dimitri/doxygen/index.html, Accessed 08 September 2016

[35] GPI–2 - Global Address Space Programming Interface, http://www.gpi-site.com accessed 08 September 2016

[36] Spiering, F.: Coupling of TAU and TRACE for parallel accurate flow simulations, International Symposium on Simulation of Wing and Nacelle Stall, Braunschweig, Germany, 2012.

[37] Spiering, F., Kelleners, P.: Coupling of Flow Solvers with Variable Accuracy of Spatial Discretization, New Results in Numerical and Experimental Fluid Mechanics IX. Contributions to the 18th STAB/DGLR Symposium, Stuttgart, Germany, 415-423, Springer Verlag, ISBN 978-3-319-03157-6, 2012.

[38] Hartmann, R., Leicht, T.: Generalized adjoint consistent treatment of wall boundary conditions for compressible flows, J. Comput. Phys., Vol. 300, 754-778, DOI: 10.1016/j.jcp.2015.07.042, 2015

[39] Hartmann, R., Leicht, T.: Generation of unstructured curvilinear grids and high-order Discontinuous Galerkin discretization applied to a 3D high-lift configuration, Int. J. Num. Meth. Fluids, Published online. DOI: 10.1002/fld.4219, 2016

[40] DIN EN ISO 9001:2015: Quality management systems – Requirements

[41] Eigen - A C++ template library for linear algebra, http://eigen.tuxfamily.org/, Accessed 08 September 2016

[42] Barth, D.J., Jespersen, D.C.: The Design and Application of Upwind Schemes on Unstructured Meshes. AIAA Paper 89-0366, 1989

[43] Venkatakrishnan, V.: On the Accuracy of Limiters and Convergence to Steady State Solutions. AIAA Paper 93-0880, 1993

[44] Schwöppe, A., Diskin, B.: Accuracy of the Cell-Centered Grid Metric in the DLR TAU-Code. In: New Results in Numerical and Experimental Fluid Mechanics VIII Notes on Numerical Fluid Mechanics and Multidisciplinary Design, 121. Springer Verlag. pp. 429-437. ISBN 978-3-642-35679-7. ISSN 1612-2909, 2013

[45] 5th AIAA CFD Drag Prediction Workshop, New Orleans, LA, 23-24 June 2012, https://aiaa-dpw.larc.nasa.gov/Workshop5/workshop5.html, Accessed 08 September 2016

[46] Bassi, F., Rebay, S.; A high-order accurate discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations, Journal of Computational Physics 131, 267-279, 1997

[47] Bassi, F., Rebay, S., Mariotti, G., Pedinotti, Savini, S.: A high-order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows, in M. Decuypere, R. & Dibelius, G. (eds.): 2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, Antwerpen, Belgium, March 5-7, 99-108, 1997

[48] Mulder, W.A., an Leer, B.: Experiments with implicit upwind methods for the Euler equations, Journal of Computational Physics 59, 232–246, 1985

[49] CASE-2 - SGI ICE X, Intel Xeon E5-2695v2 12C 2.400GHz, Infiniband FDR, https://www.top500.org/system/178196, Accessed 08 September 2016

[50] Eisfeld, B., Rumsey, C., Togiti, V.: Verification and Validation of a Second-Moment-Closure Model, AIAA Journal, Vol. 54, No. 5, 1524-1541, DOI: 10.2514/1.J054718, 2016

[51] Togiti, V., Eisfeld, B.: Assessment of g-Equation Formulation for a Second-Moment Reynolds Stress Turbulence Model, AIAA 2015-2925, AIAA Aviation, 45th AIAA Fluid Dynamics Conference, 22. - 26. June, Dallas, Texas, USA, 2015

[52] Menter, F.R., Langtry, R.B.: Correlation-Based Transition Modeling for Unstructured Parallelized Computational Fluid Dynamics Codes, AIAA Journal, Vol. 47, No. 12, 2894-2906, DOI: 10.2514/1.42362, 2009

[53] Grabe, C., Nie, S., Krumbein, A.: Transition Transport Modeling for the Prediction of Crossflow Transition, AIAA 2016-3572, AIAA Aviation and Aeronautics Forum and Exposition, 13.-17. June, Washington, DC, USA, 2016