IX International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2021
E. Oñate, M. Papadrakakis and B. Schrefler (Eds)

ACCELERATING THE FLOWSIMULATOR: PROFILING AND SCALABILITY ANALYSIS OF AN INDUSTRIAL CFD-CSM TOOLCHAIN

IMMO HUISMANN*, LARS REIMER†, SEVERIN STROBL*, JAN EICHSTÄDT*, RONNY TSCHÜTER*, ARNE REMPKE*, AND GUNNAR EINARSSON†

*Institute of Software Methods for Product Virtualization
German Aerospace Center (DLR), Zwickauer Straße 46, 01069 Dresden, Germany
e-mail: immo.huismann@dlr.de, web page: `https://www.dlr.de`

†Institute of Aerodynamics and Flow Technology
German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Brunswick, Germany

Key words: CFD, CSM, Scalability Analysis, High-Performance Computing

Abstract. Aeroelasticity simulations increase in importance for aircraft design, requiring an efficient coupling of computational fluid dynamics (CFD) with computational structure mechanics (CSM) solvers. This contribution investigates the scalability of a high-fidelity CFD-CSM toolchain on modern high-performance computing (HPC) architectures. It consists of DLR's `TAU` solver for fluid dynamics simulations [1], and `FlowSimulator` [2] components for the incorporation of precomputed structural normal mode data, as well as for the underlying mesh deformations. The computational performance of the entire simulation pipeline is evaluated using a single measurement suite, allowing to identify bottlenecks of individual components and differences in their scalability. Preliminary improvements are realized via hybrid parallelization. Although this study focuses on a specific toolchain, key findings about scalability issues are relevant for complex CFD-CSM or other coupled simulations in general.

1 Introduction

Drastic reductions in CO_2 and NO_x emissions are an important goal in the design of future aircraft, rendering accurate and fast simulations of radically new aircraft concepts more relevant than ever.

While rapid development of aircraft requires simulations of these, a fully resolved simulation of an aircraft is projected to become feasible only in 2030 [3]. Reaching this target requires advances in both hardware and software. As a first aspect, utilizable compute

power and memory bandwidth of hardware components has to improve. As a second aspect, enhancements in the software design are necessary to increase scalability and node-level performance [4]. Furthermore, novel algorithms, for instance high-order methods such as the continuous and discontinuous spectral-element methods, and reduced-cost models, e.g. coupling of REYNOLDS-averaged NAVIER-STOKES equations (RANS) with large-eddy simulations, offer the potential to improve the quality of simulations.

From the software side, domain decomposition has carried us into and throughout the petascale era, however it is not sufficient for fully leveraging today's high-performance computers [5]. To achieve higher scalability, most toolchains will need to incorporate at least hybrid parallelism, such as `MPI + OpenMP`. Yet, these efforts are mostly contained to flow and structure solver, with the remainder of the coupling toolchains using at most domain decomposition. While sufficient for low numbers of nodes with only a few cores each, as typical of the late 2000s, HPC systems have moved on to thousands if not tens of thousands of highly parallel nodes, straining the toolchains and laying bare their least scalable components.

This paper considers the flow simulation and aircraft analysis pipelines maintained by DLR which are based on the `FlowSimulator` environment first discussed in [6]. `FlowSimulator` establishes a programming interface for in-memory data exchange between codes and a common user interface. This allows for pipelines that range from flow simulations [7], over simulations for multi-disciplinary analysis [2], to shape and structure optimization of aircraft [8]. Standard pipelines for such scenarios are investigated with regard to their scalability. In the considered cases, all components of the pipelines, from import and export over repartitioning to flow solution and coupling with structure solvers, are investigated with regard to their performance and impact on the scalability of the whole pipeline.

2 The FlowSimulator environment

2.1 Overview

The `FlowSimulator` started as an effort by Airbus and DLR for a unified environment for parallel flow simulation capabilities in 2005. The `FlowSimulator DataManager` (`FSDM`) library lays the foundation for the `FlowSimulator` environment. `FSDM` provides mesh-oriented data structures for in-memory coupling of `FlowSimulator` components, such as distributed multi-dimensional arrays, meshes, datasets, and methods for communication of these. Furthermore, `FSDM` implements basic functionalities such as mesh import, export and partitioning for both structured and unstructured meshes, handling of geometry and boundary conditions, storing of datasets and dimensionalization thereof. While most of these operations are not compute-intensive, the majority is memory-intensive and requires unstructured data accesses. Hence, in order to maximize the efficiency on HPC systems, `FSDM` is implemented in C++, with mesh data structures forming the very core of the parallelization concept for distributed memory systems. The handling of arbitrary

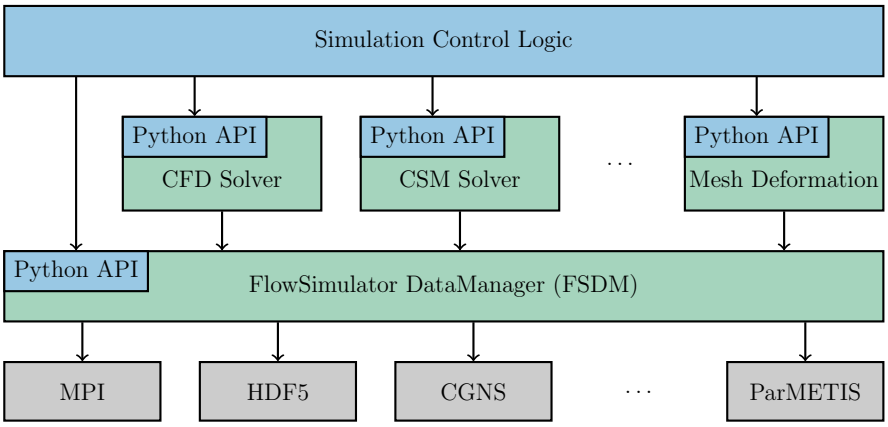

Figure 1: Interaction of `FlowSimulator` components. Green parts indicate C++ libraries, blue parts `Python` interfaces, and gray boxes external libraries.

datasets attached to nodes or cells as well as various mesh operations are based on these data structures.

In `FSDM` a small hierarchy of mesh-based data structures exists: An instance of the data manager can store multiple meshes and each mesh can contain multiple datasets that are either node- or cell-based. A factory pattern allows plugins to share the same generic interface as `FSDM` itself for operations on meshes. Moreover, plugins can load and also create new datasets attached to meshes, for instance flow solutions, pressure fields or deformed coordinates, allowing another component to read and modify these in a second step. Lastly, by storing multiple meshes within a single data manager, different simulation codes can be coupled. For instance a specialized aerodynamics code can solve for the exterior flow around an aircraft while a highly specialized code computes the interior of the turbine cascades.

All higher-level operations of `FSDM` and plugins are wrapped to `Python` using `SWIG`. This allows the control logic to be implemented in an easily-accessible language, while performance-critical parts benefit from the computational efficiency of the C++ layer, as shown in Figure 1. The usage of `Python` buffer objects allows for the exchange of data between the C++ and the `Python` layer without incurring the cost of an additional copy. Thus not only developers but also simulation experts can write scenario scripts and easily change and adapt the simulation pipelines to their needs.

Multiple high-fidelity flow solvers integrate into the `FlowSimulator`, such as `DLR TAU` [1], the flow solver "CFD for ONERA, DLR and Airbus" (`CODA`) [7] that is being jointly developed based on DLR's `Flucs` [9], and lastly `TRACE`. Furthermore, an interface for coupling with `HYDRA` is under development. From the structural mechanics side, `FlowSimulator` plugins for `Nastran` and `Ansys` exist. These codes are, however, proprietary closed-source software that cannot easily be interfaced with `FSDM`. For each interaction or data transfer, all input files, i.e. meshes along with the relevant fields, need to be written to and read from disk. To circumvent this restriction, a plugin relying solely on in-memory data transfers is being implemented for DLR's `B2000++`. For implementation of PDE solvers,

`FSPETSc` wraps linear and non-linear solvers from `PETSc` and also those from the new linear solver library `Spliss` [10].

Not all scenarios can rely on the mesh being constant over time. For instance fluid-structure coupling leads to displacements of the surface CFD mesh, and in turn a different volume mesh. Similarly, trimming of flaps changes the boundary geometry of the mesh. Generating valid meshes for each simulation step by re-meshing the flow domain would be costly and not well parallelizable. Therefore, deforming a given volume mesh (i.e. from the previous simulation step) is preferred in `FlowSimulator` pipelines and multiple plugins exist for mesh deformation and repair. For instance, `FSDeformation` implements mesh deformation based on radial-basis functions [11] and `FSMeshDeformation` mesh deformation based on the elastic analogy [12], using `FSPETSc` as the underlying solver.

Existing control layer implementations of Airbus and DLR implement pipelines ranging from standard CFD pipelines over simulations of maneuvers and fluid-structure interaction, e.g., static aeroelastic coupling [2], to gradient-based optimization pipelines for aircraft design [8]. The latter is being consolidated using OpenMDAO as a top layer for `FlowSimulator` for shape optimization [13].

The `FlowSimulator` itself is proprietary, but the `FlowSimulator DataManager` is distributed under an open source license (LGPL).[1]

2.2 Parallelization

The `FlowSimulator` is intended to be a user-friendly platform. Thus, accessibility for the end user is a key priority for its development. While most users are well-versed in their application domain, specifics of parallel programming typically are not part of their daily concerns. Therefore, the parallelization should be hidden from the user; the run scripts should not contain details of the parallelization itself. As the amount of time spent in the `Python` scenario code remains minimal, by delegating the performance and scalability critical parts to C++ routines, an overall efficient code is obtained.

With flow simulations in mind, the memory available on one compute node is seldom sufficient for containing the whole mesh. Hence, a domain decomposition splits the computational domain into smaller subdomains, which are distributed among the processes, leading to a distributed mesh and lower memory requirement per process. Mesh data structures of `FSDM` implement domain decomposition for structured and unstructured meshes using `MPI`. Plugins retain the decomposition and resume working on process-local data.

Scalability is further improved by employing two-level parallelism: A coarse-level implements domain decomposition via `MPI`, while inside the domains data parallelism is exploited, either via threads using `OpenMP`, accelerator cards or both. Increasing the number of threads per process while lowering the number of processes allows for the par-

[1] `FlowSimulator DataManager` is available at `https://gitlab.as.dlr.de/FSDM/FSDM` after registration.

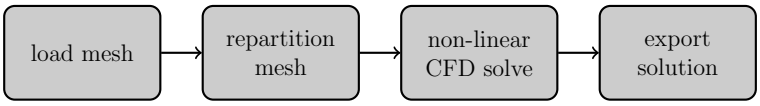

Figure 2: Flow chart for a CFD computation using `FlowSimulator`.

titions on the coarse level to grow, leading to a lower volume in communication on the `MPI` layer and a higher scalability. For example, the flow solver `CODA` uses `OpenMP` to manage its thread pools while implementing its own work-sharing, resulting in a significant gain in scalability over the pure `MPI`-parallelized `DLR TAU` [5].

3 Typical FlowSimulator toolchains

3.1 Flow simulations

Flow simulations are the bread and butter use case of the `FlowSimulator`. The applications range from single case steady-state simulations over predefined parameter studies, such as computation of drag polars and flight envelopes, to unsteady simulations for maneuvers. Compared to a monolithic flow solver, the `Python` user interface allows to run complete simulation chains for pre-defined parameter settings while automatically updating simulation parameters, for instance angle of attack, flap configurations, or other boundary conditions. This enables scenarios like reacting to probe data obtained from field or surface solution, changing simulation parameters mentioned above, and restarting the solver.

In this work, a standard CFD pipeline – simulating the flow around an aircraft configuration – was chosen to demonstrate the usage of the `FlowSimulator` (compare Figure 2). The pipeline implements the Common Research Model (CRM) testcase from the drag prediction workshop of NASA [14]: A steady-state flow around the wing-body configuration based on the RANS equations is to be computed at an upstream Mach number of $Ma = 0.85$ and angle of attack of $\alpha = 2.209°$. The Reynolds number is $Re = 50 \cdot 10^6$, based on a Reynolds length of $Re_\mathrm{L} = 275.8$ inches. All further parameters were chosen in accordance with [1] to facilitate an easy comparison. Multiple grid series exist for the CRM; here, the unstructured meshes from [14] are utilized. These are based on a block-structured half-body grid series and subdivide the hexahedral into prismatic elements. The flow solver `CODA` at version 2021.03 is solving the CRM case using a finite volume scheme. A two-stage simulation is utilized; using the free-stream conditions as the initial condition, the first stage does not use the gradients for the reconstruction of the state, leading to a higher stability, while the second stage incorporates the gradients into the reconstruction. On the first stage, the initial residual is reduced by a factor of 10^{-5}, the second stage solves to a relative residual reduction of 10^{-10}.

Figure 3 depicts the resulting convergence history for the first grid of the series. Only 24 iterations suffice for the first stage. With the gradients being incorporated into the reconstruction, the residual jumps and further 40 iterations are required. A comparison of the resulting drag coefficient C_D between `DLR TAU` and `CODA` is shown as well in Figure 3.

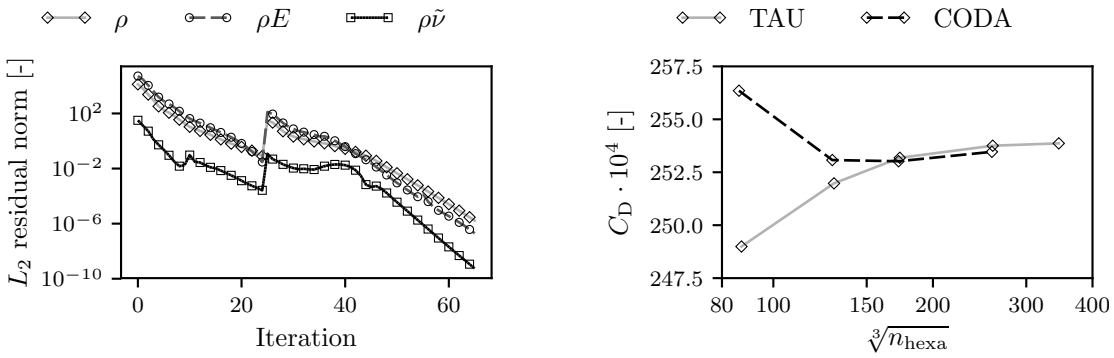

Figure 3: Left: Convergence history on the "tiny" block-structured grid with `CODA` and right: drag coefficient C_D comparing results obtained with `DLR TAU` and `CODA`.

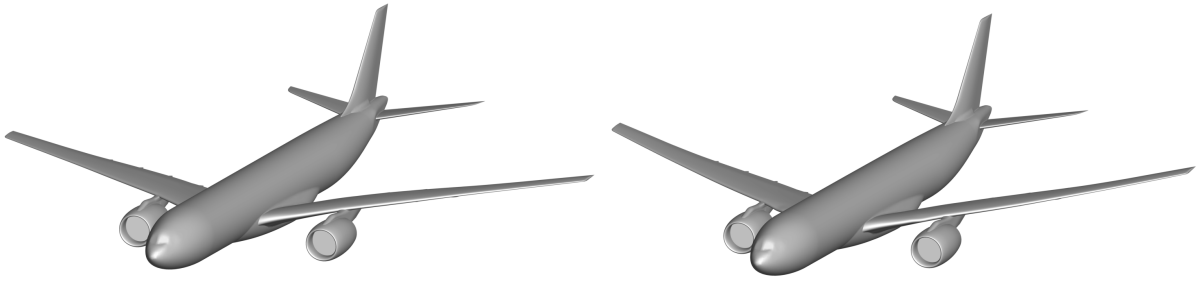

Figure 4: XRF1 configuration. Left: undeformed state, right: aeroelastic equilibrium.

While the difference is large on the first grid, less than a percent difference remains for the finest grid from [14].

3.2 Static aeroelastic coupling

A second more elaborate testcase considers the static aeroelastic simulation of the flow around a passenger aircraft configuration, including powered engines, called XRF1, as shown in Figure 4. It is an Airbus provided industrial-standard research testcase representing a typical configuration for a long-range aircraft and is used to engage with external partners on development and demonstration of relevant capabilities and technologies.

The fully turbulent flow around the aircraft is modeled using the RANS equations, while the structure of the aircraft is modeled with linear elasticity for both, material model and geometry. The upstream Mach number is chosen as $Ma = 0.85$, the Reynolds number as $Re = 60.7 \cdot 10^6$ based on the mean aerodynamic chord length of 9.096 m, and the angle of attack as $\alpha = 2°$. `DLR TAU` serves as the flow solver and the Spalart –Allmaras turbulence model in negative formulation for computation of the Reynolds stresses [15]. The CFD mesh consists of $104.8 \cdot 10^6$ cells: $38.6 \cdot 10^6$ prisms in the boundary layer mesh and $65.4 \cdot 10^6$ tetrahedra in the volume mesh. A parallel modal CSM solver is employed to compute the structural deformation. Modeshapes and eigenfrequencies of the unrestrained (free-free) structural model are computed in an offline preprocessing step using `Nastran`. The lowest 30 elastic modes are considered during aeroelastic coupling.

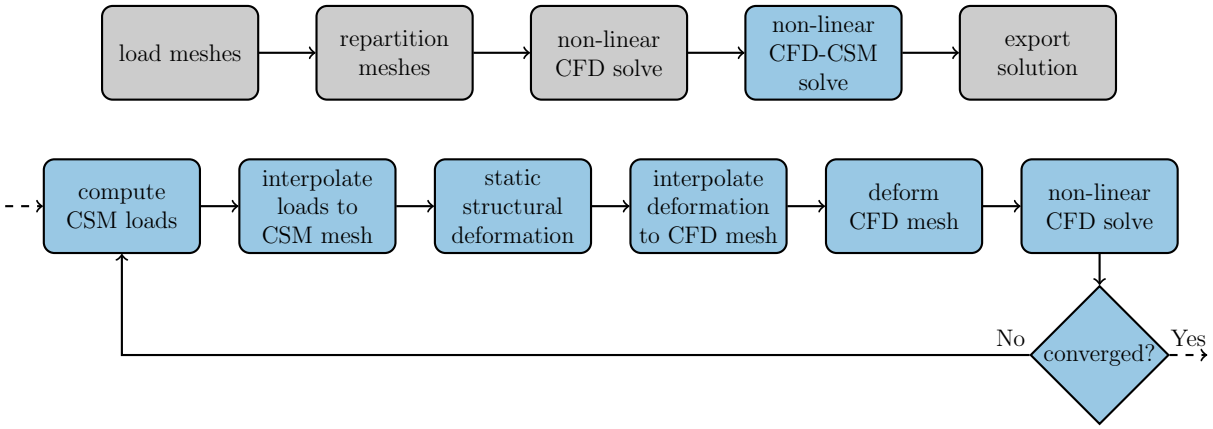

Figure 5: Flow chart for a CFD-CSM coupling in `FlowSimulator`, the overall pipeline is shown on the top and the CFD-CSM coupling on the bottom.

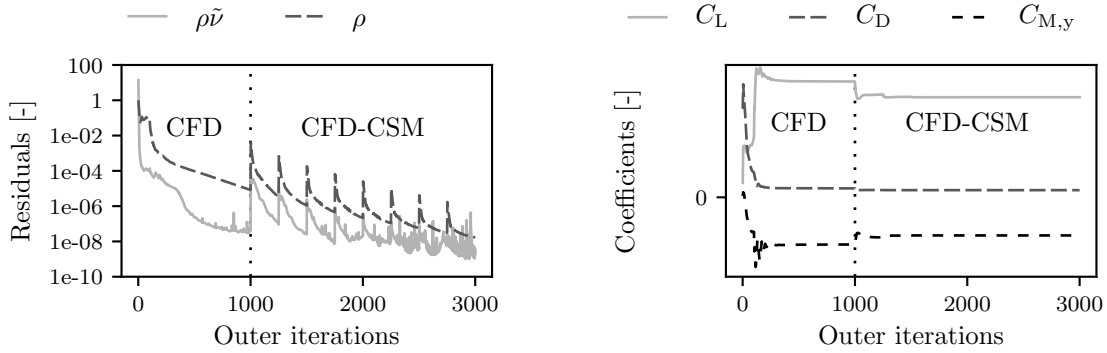

Figure 6: Results for the XRF1 testcase. Left: History of the residuals in density ρ and turbulence variable $\rho\tilde{\nu}$, right: coefficients for lift C_L, drag C_D, and pitch momentum $C_\mathrm{M,y}$.

Figure 5 depicts the flow chart of the resulting pipeline. As before, `FSDM` handles mesh import and repartitioning, this time for both CFD and CSM meshes. Then, the initial CFD solution is computed as a third step, treating the airframe as a rigid body by using the original CFD mesh. A Gauss-Seidel method implements the CFD-CSM coupling: The current aerodynamic loads are projected onto the structure mesh using the Finite Interpolation Elements (FIE) approach [16]. Then, the same algorithm interpolates the structural deformation to the CFD surface mesh using an under-relaxation factor of 0.7, while respecting the principle of preserving virtual work. Subsequently, a mesh deformation based on radial basis functions (RBF) [11] adapts the CFD volume mesh to the deformed CFD surface. In the case of ill-conditioned or even inverted cells, a mesh deformation based on an elastic analogy as described in reference [12, 2] repairs the mesh. Lastly, a CFD solution is computed on the new mesh. The process repeats until changes in displacement and CFD residuals lie below 10^{-6}.

Figure 6 depicts the convergence history of the CFD solver. The initial CFD solution is computed using 1000 outer iterations of the non-linear multigrid solver of `DLR TAU`. This proves sufficient for reducing the initial residual by a factor of 10^{-5} and 10^{-8} for density

and the turbulent viscosity, respectively. Thereafter the CFD-CSM coupling starts. The aerodynamic loads result in a deformation of the aircraft, which feeds back into the CFD mesh by changing the geometry of the boundary. This in turn leads to a jump in the residuals and a further 250 outer iterations per coupling step are performed in the CFD simulation. After eight coupling steps, the residual of the CFD solver remains below 10^{-7} of the initial one.

Figure 4 depicts the resulting deformation, which results in an upward bending of the wings. Figure 6 shows the history of drag coefficient C_{D}, lift coefficient C_{L}, and pitch momentum coefficient $C_{\mathrm{M,y}}$ over the course of the computation. The lift reduction in aerodynamic equilibrium compared to the undeformed case is observable from the lift coefficient, which is typical for backswept wings. It results from the bending-twist coupling and the associated spanwise local reductions of angle of attack.

4 Profiling the FlowSimulator

4.1 Computational setup

All performance measurements were executed on the high-performance computer Cara of DLR. On Cara, one node consists of two sockets filled with one AMD EPYC 7601 CPU, each with 32 cores. Both CPUs ran on a clock frequency fixed to 2200 MHz, such that frequency scaling does not impede the measurements. The `FlowSimulator` was compiled using GCC v. 8.2.0 with flags `-O3 -march=native`.

The pipelines presented in Section 3 were run again, scaling the number of nodes from $1 = 2^0$ to $64 = 2^6$ and, in turn, the number of cores from $n_{\mathrm{cores}} = 64$ to $n_{\mathrm{cores}} = 4096$. The runtime contributions of the toolchain components were measured and speedup and parallel efficiency computed. For the latter two, one full node served as reference.

4.2 CFD simulations

Runtime measurements for the CRM testcase from 3.1 were performed using one thread per process on the "fine" mesh containing 1.3 million prism elements. The expectation for this testcase is that the CFD solver takes up the biggest portion of the runtime while runtime contributions of other parts of the pipeline remain negligible.

Figure 7 depicts the runtimes, speedups, and resulting parallel efficiencies for each of the substeps of the simulation. As expected, the CFD solver is the most expensive step of the pipeline: On one node, this step's runtime is about a factor of 100 longer than the next step, the initialization of the CFD solver. But with nearly 100 % parallel efficiency up until 8 nodes, and 90 % and 70 % when using 16 and 32 nodes, respectively, it scales satisfactorily.

The second biggest runtime contribution stems from the initialization of `CODA`, which requires an expensive wall distance computation, but scales well overall. The mesh repartitioning and import consume less than one percent of the runtime when using one node. Their runtime, however, does not decrease with an increased core count; it increases. This

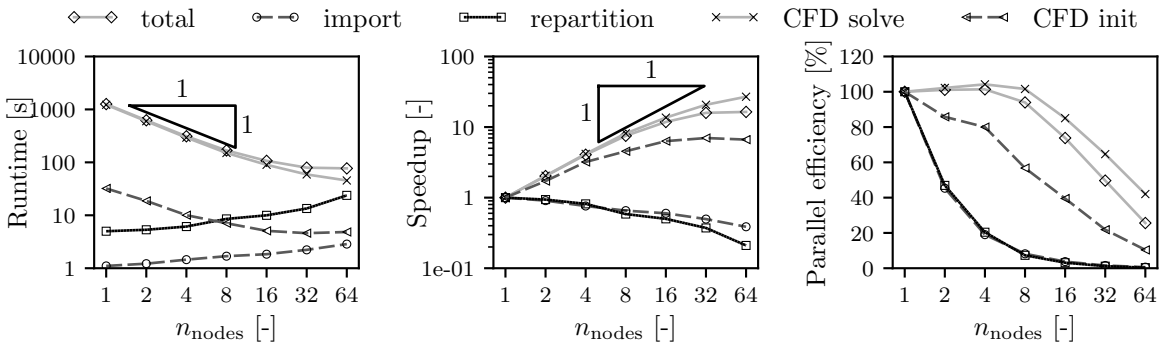

Figure 7: Results for the CRM testcase using `CODA` with one thread per process.

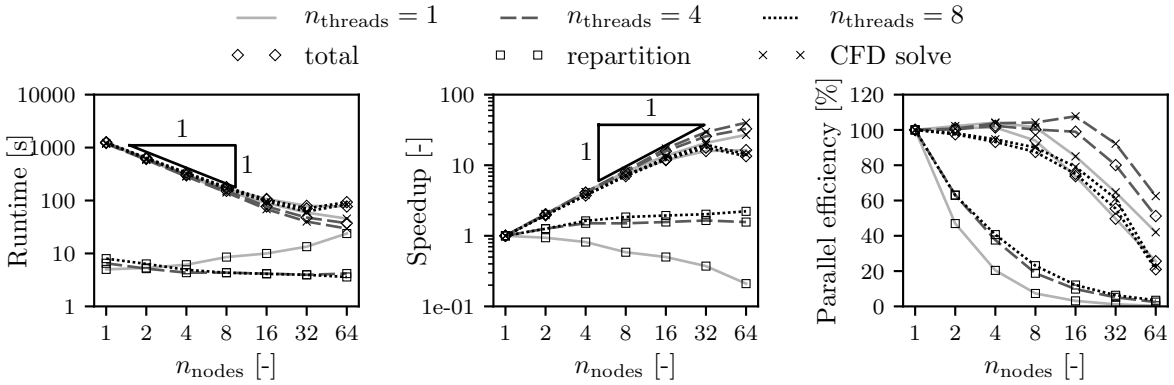

Figure 8: Results for the CRM testcase using `CODA` with 1, 4, and 8 threads per process.

leads to the repartitioning taking more time than the flow solver initialization at 8 nodes and competing with the runtime of the flow solver at 64, severely impacting the scalability of the overall pipeline: While `CODA` still exhibits 70 % parallel efficiency on 16 nodes, the overall efficiency then lies near 50 %.

The lack of scalability of import and repartitioning are inherent to the implementation: The import reads chunks of the mesh and directly sends them to other processes, leading to a lower memory consumption for the master process. While this circumvents memory restrictions, it leads to a sub-optimal partitioning that is sent into a graph partitioner and the whole mesh is redistributed. The higher amount of computation for the graph partitioner and communication thereafter culminate in a higher runtime, in the present case a third of the total runtime. In effect, the parallel efficiency plummets.

The typical way to increase scalability lies in using two-level parallelism. However, of the discussed CFD pipeline, only `CODA` itself implements hybrid parallelization, while all other components do not. Using fewer and larger partitions can still generate a performance benefit for import and export: As the runtime grows with the number of processes, using fewer of these postpones the runtime issues encountered at large numbers of cores.

The testcase was repeated using 4 and 8 threads, i.e. one process per L3 cache partition and one process per NUMA domain, with Figure 8 depicting the runtimes and resulting speedups and scalabilities. While no runtime benefit exists on one node, using hybrid

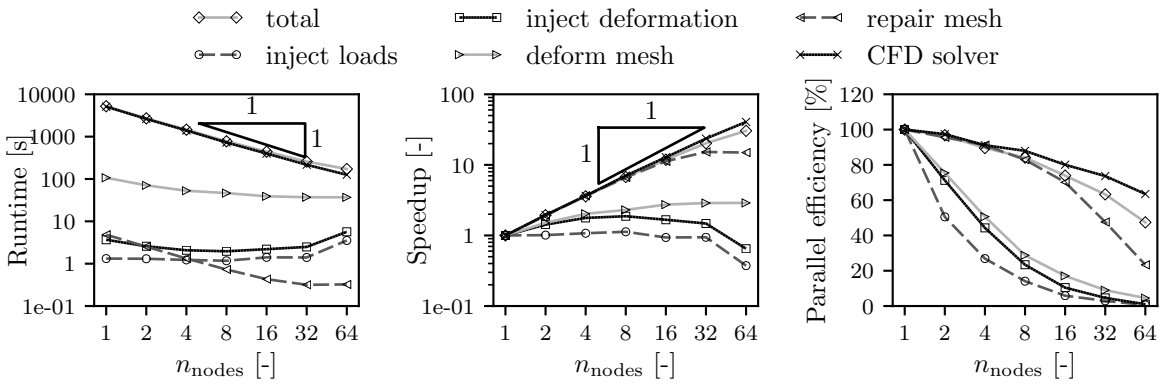

Figure 9: Runtimes per coupling step and derived quantities for the XRF1 testcase.

parallelism proves beneficial: With 4 threads per process, it extends the scalability of the flow solver. On 16 nodes, >100 % parallel efficiency results and >90 % on 32 nodes, which corresponds to ≈ 640 elements per core. Using 8 threads, does not generate a lower runtime than the single-threaded case. The issue can be traced back to the architecture: While the memory interface is shared between two 4 core packages, the L3 cache is not. The threads do not share data over the L3 cache partitions, but over the memory interface.

Benefits for the flow solver were to be expected, but the repartitioning also exhibits performance gains. As it does not contain any thread-level parallelism, these gains result counter-intuitively from using fewer processes, lowering the volume of performed computation and communication. Furthermore, the lower number of processes per node leads to higher available cache, memory, and network bandwidth for the remaining processes. The combination of both stabilizes the runtime for the repartitioning, and therefore a lower impact on the scalability of the pipeline. Even at 32 nodes, the repartitioning now uses less than a tenth of the runtime of the flow solver instead of half, leading to a parallel efficiency of the pipeline of 80 % at only ≈ 640 elements per core.

4.3 Static aeroelastic coupling

Figure 9 shows the runtimes per coupling step for the components of the static aeroelastic coupling pipeline from 3.2. Mesh import and repartitioning were excluded to focus on the interior of the pipeline. On one node, the flow solver takes the highest amount of runtime with 4000 s, nearly 40 times more than the next contribution. The only other noteworthy part of the pipeline, the RBF-based deformation of the CFD volume mesh using 100 s, all other parts of the pipeline require well below 10 s.

As before, the CFD solver scales well, retaining 80 % parallel efficiency at 16 nodes and near 70 % when using 64 nodes. The overall pipeline, however, exhibits a parallel efficiency of 75 % at 16 nodes and only 50 % on 64 nodes. This stems from the low scalability of the mesh deformation. It exhibits an asymptotic runtime near 35 s, slowly converging towards a speedup of 3, and is a prime example of Amdahl's law: The solution process itself is not sufficiently parallelized, leading to 4095 out of 4096 processes waiting for a

significant amount of time for one process. The RBF-based deformation uses far fewer control points and corresponding basis functions to describe the deformation than there are points in the full CFD mesh. This approach reduces the computational cost to a practical amount, but incurs solving a dense linear system. The latter is not parallelized and, therefore, does not scale well. Furthermore, reducing the data from the full mesh to the control points in order to prepare the core problem and communicating the solution back to all processes constitutes a communication bottleneck. Only the application of the weighted RBF functions on the mesh points and a local nearest-neighbor-correction is done in parallel and can therefore scale. As before with the pure CFD pipeline, parts of this pipeline that start out as seemingly inconsequential lead to scalability issues for > 500 cores.

5 Conclusions

This paper conducted a performance analysis of two industry-grade toolchains using the `FlowSimulator`, one performing a pure CFD simulation and one simulating the static aeroelastic coupling of a long-haul passenger aircraft. For the CFD toolchain, the flow solver `CODA` scales near perfectly, achieving a high parallel efficiency even for modest numbers of elements per node, while the mesh import from `FSDM` and graph-based partitioning functions do not. This leads to the latter taking up 30 % of the runtime for high node counts and ultimately dominating the overall runtime. A similar effect can be observed for the second testcase, where the flow solver `DLR TAU` exhibits excellent scaling characteristics, whereas the (not fully parallelized) mesh deformation using radial basis functions [11] does not. In both cases, components which, for moderate numbers of nodes behave inconspicuous, end up limiting the scalability of entire pipelines in strong scaling scenarios.

This illustrates the need to examine the scaling behavior not only of the individual components of a simulation pipeline, but rather all parts combined into a realistic simulation scenario. Estimating the contributions to the total runtime a priori for different configurations and in dependence on the available computational resources is extremely challenging. Thus performance evaluations not merely of high-profile candidates such as CFD codes, but also the often overlooked coupling or pre-processing routines are crucial. In the case of the `FlowSimulator`, this allowed us to identify several critical (yet localized) sections where small changes can result in clear improvements in the overall scalability. While for newly developed codes it is common practice to use multi-layered parallelization schemes, it can also be worthwhile to integrate such schemes into legacy codes, as demonstrated here.

Acknowledgements The authors would like to thank Airbus for providing the XRF1 testcase as a mechanism to demonstrate the approaches presented in this paper. Furthermore, the authors would like to thank their colleagues Axel Schwöppe and Ralf Hartmann for the fruitful discussions on the solver parameter choices.

REFERENCES

[1] S. Langer, A. Schwöppe, and N. Kroll, "Investigation and comparison of implicit smoothers applied in agglomeration multigrid," *AIAA Journal*, vol. 53, no. 8, pp. 2080–2096, 2015.

[2] L. Reimer, R. Heinrich, S. Geisbauer, T. Leicht, S. Görtz, M. R. Ritter, and A. Krumbein, "Virtual aircraft technology integration platform: Ingredients for multidisciplinary simulation and virtual flight testing," in *AIAA SciTech Forum*, January 2021.

[3] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, and D. Mavriplis, "CFD vision 2030 study: a path to revolutionary computational aerosciences," 2014.

[4] A. Probst, T. Knopp, C. Grabe, and J. Jägersküpper, "HPC requirements of high-fidelity flow simulations for aerodynamic applications," in *Euro-Par 2019: Parallel Processing Workshops*, (Cham), pp. 375–387, Springer International Publishing, 2020.

[5] N. Kroll, M. Abu-Zurayk, D. Dimitrov, T. Franz, T. Führer, T. Gerhold, S. Görtz, R. Heinrich, C. Ilic, J. Jepsen, *et al.*, "DLR project Digital-X: towards virtual aircraft design and flight testing based on high-fidelity methods," *CEAS Aeronautical Journal*, vol. 7, no. 1, pp. 3–27, 2016.

[6] M. Meinel and G. O. Einarsson, "The FlowSimulator framework for massively parallel CFD applications," in *PARA2010*, Juni 2010.

[7] I. Huismann, S. Fechter, and T. Leicht, "HyperCODA – extension of CODA flow solver towards hypersonic flows," 2020. accepted.

[8] S. Görtz, M. Abu-Zurayk, C. Ilic, T. F. Wunderlich, S. Keye, M. Schulze, C. Kaiser, T. Klimmek, Ö. Süelözgen, T. Kier, *et al.*, "Overview of collaborative multi-fidelity multidisciplinary design optimization activities in the DLR project VicToria," in *AIAA Aviation 2020 Forum*, p. 3167, 2020.

[9] T. Leicht, J. Jägersküpper, D. Vollmer, A. Schwöppe, R. Hartmann, J. Fiedler, and T. Schlauch, "DLR-project Digital-X – next generation CFD solver 'Flucs'," in *Deutscher Luft- und Raumfahrtkongress 2016*, 2016.

[10] O. Krzikalla, A. Rempke, A. Bleh, M. Wagner, and T. Gerhold, "Spliss: A sparse linear system solver for transparent integration of emerging HPC technologies into CFD solvers and applications," 2020. submitted.

[11] R. Heinrich, N. Kroll, W. Krueger, and B. Nagel, "Fluid-structure coupling for aerodynamic analysis and design: A DLR perspective," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, 2008.

[12] B. Stickan, A. Rempke, S. Helm, and H. Bleecke, "High-fidelity CFD-CSM interaction in the industrial context," in *IFASD 2017-International Forum on Aeroelasticity and Structural Dynamics*, 2017.

[13] T. Backhaus, S. Gottfried, A. Merle, J. T. Hwang, and A. Stueck, "Modularization of high-fidelity static aeroelastic MDO enabling a framework-based optimization approach for HPC," in *AIAA Scitech 2021 Forum*, p. 1236, 2021.

[14] NASA, "AIAA CFD 5th drag prediction workshop." `https://aiaa-dpw.larc.nasa.gov/Workshop5/test_cases_5.htm`.

[15] S. R. Allmaras and F. T. Johnson, "Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model," in *Seventh international conference on computational fluid dynamics (ICCFD7)*, pp. 1–11, 2012.

[16] A. Beckert, "Coupling fluid (CFD) and structural (FE) models using finite interpolation elements," *Aerospace Science and Technology*, vol. 4, no. 1, pp. 13–22, 2000.