Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Fluid-Structure Simulations with OpenFOAM for Aircraft Designs

Thomas Ponweiser[a], Peter Stadelmeyer[a,]*, Tomáš Karásek[b]

[a] *Johannes Kepler University Linz, RISC Software GmbH, Austria*
[b] *VŠB-Technical University of Ostrava, IT4Innovations, Czech Republic*

Abstract

Multi-physics, high-fidelity simulations become an increasingly important part of industrial design processes. Simulations of fluid-structure interactions (FSI) are of great practical significance – especially within the aeronautics industry – and because of their complexity they require huge computational resources. On the basis of OpenFOAM a partitioned, strongly coupled solver for transient FSI simulations with independent meshes for the fluid and solid domains has been implemented. Using two different kinds of model sets, a geometrically simple 3D beam with quadratic cross section and a geometrically complex aircraft configuration, runtime and scalability characteristics are investigated. By modifying the implementation of OpenFOAM's inter-processor communication the scalability limit could be increased by one order of magnitude (from below 512 to above 4096 processes) for a model with 61 million cells.

Application code OpenFOAM, Computational Fluid Dynamics, Computational Structural Mechanics, Fluid-Structure Interaction

1. Introduction

One of the main driving forces within the aeronautics industry is the continuous need to design aircrafts with progressively reduced environmental footprints. Meeting this challenge is an essential prerequisite for public acceptance of any further growth of air traffic. Today simulation systems are a key technology for the design of lighter, more fuel-efficient, noise-reduced and at the same time safer aircrafts (see e.g. [1], p.121, Engineering Sciences and Industrial Applications). To evaluate and optimize new designs high-fidelity simulations are needed that cover multiple disciplines and accurately model the interactions between them, especially between aerodynamics and structural mechanics. The goal of this work is to demonstrate the opportunities and challenges of high-performance computing in the area of multi-physics simulations, particularly for fluid-structure interactions (FSI).

On the basis of OpenFOAM a partitioned, strongly coupled solver for transient fluid-structure simulations with independent meshes for the fluid and solid domains has been implemented. Two different sets of test cases have been created for analysing the runtime performance with respect to parallel scalability. Overall scalability of the coupled solver has been investigated using a solid 3D beam with quadratic cross section, which is fixed at one end. Profiling using HPCToolkit showed that the computation of mesh deformations of the fluid domain, as currently implemented in OpenFOAM, is a major limiting factor. By changing OpenFOAM's inter-processor communications stream implementation the scalability limit could be increased by one order of magnitude (from below 512 to above 4096 processes). The second test case models a full aircraft configuration and is intended as a representative example for an industrial application. This example was used to explore the scalability behaviour of

* Corresponding author. *E-mail address*: peter.stadelmeyer@risc.jku.at

the individual processing and computations steps that are part of a typical industrial use case including initial mesh generation.

The paper is structured in the following way: In Section 2 we briefly describe the underlying scientific case and the initial situation of our work. The implementation of the solution algorithm for the coupled simulation and the system installation that was used for performance analysis is described in Section 3. Results of scalability tests for the coupled solver using the 3D beam example are shown in Section 4. In this section we also explain the scalability bottleneck that is most relevant for our application and we present an alternative implementation for it. In Section 5 runtime and scalability results for the individual computation steps using a complete aircraft model are discussed. A summary of our findings and an outlook for further development is given in Section 6.

2. Objectives of work

2.1. Scientific case

During the conceptual and preliminary design phases the basic shape and hence the fundamental aerodynamic properties of aircrafts are defined. From such an initial design estimates about aerodynamic forces acting on the aircraft structure can be derived. These forces are one of the principal factors for defining the structural design. This design subsequently determines the elastic deformation with respect to aerodynamic forces and therefore has a direct influence on aerodynamic properties. In a first design step it can be assumed that the aircraft structure is completely rigid and vice versa that the aerodynamic forces are fixed. But for more realistic simulations, especially for highly flexible aircraft designs that are exceedingly tailored for minimal drag and maximum fuel-efficiency, it is essential to consider interactions between aerodynamics and structural mechanics as an integral part of the whole design process. Therefore, a seamless combination of aerodynamic and structural design within the global design process of aircrafts has significant importance for the aeronautical industry.

For analysing and optimizing aircraft structures, simplified aerodynamic models, as for example models based on potential flow theory, have been and still are commonly used in the design process, because the computational complexity of solving Navier-Stokes equations is considerably higher. These simplified models have inherent limitations regarding their applicability and the quality of their simulation results especially for complex geometric configurations.

With this work we want to contribute an example how HPC can serve as a key technology for accomplishing more reliable and more detailed simulations as part of the design process. This includes that simulation runtimes are compatible with the high-level multidisciplinary aircraft design process and qualities of simulations are appropriate for product design.

2.2. Application code

OpenFOAM (Open Field Operation and Manipulation) is a free, open-source, general purpose finite-volume simulation framework developed by OpenCFD Ltd at ESI Group and distributed by the OpenFOAM Foundation [L1] under the GNU General Public License. It covers a wide range of features for physical and geometrical modelling, including a variety of turbulence models and various mesh modification functionalities. Instead of being a monolithic system OpenFOAM is designed as a highly customizable library where users can implement their own solution sequences using pre-defined program blocks. OpenFOAM is written in C++ and makes intensive use of object oriented and generic programming paradigms. Parallelization is based on a domain decomposition strategy and uses MPI only. In PRACE-2IP the code was selected as one of being of high interest to industrial parterners, see [2]. For our work OpenFOAM versions 2.2.1 and 2.3.0 from ESI-OpenCFD have been used.

The solver we implemented uses a partitioned formulation for modelling the fluid-structure interaction where two independent meshes, one for the fluid domain and one for the solid domain, are coupled via a common interface surface. Both the fluid dynamics computation (CFD) and structural mechanics computation (CSM) are done using OpenFOAM. We decided to use a transient solver for incompressible, laminar flow of Newtonian fluids for the fluid part, as this allows using a simple physical model that can easily be parameterized with respect to its size, i.e. number of cells, for profiling and scalability analysis. For the solid part a transient solver for linear-elastic

small-strain deformations has been implemented. Due to the modular design of OpenFOAM, both the fluid and solid solver can be exchanged independently to other types of solvers, e.g. to add a certain turbulence model or Large Eddy Simulations to the fluid computation.

2.3. Related work

According to our knowledge no detailed results about scalability for such types of simulations with OpenFOAM in the context of HPC do exist. From previous profiling work done for OpenFOAM it is known that I/O operations and metadata handling can become a limiting factor with respect to scalability (see [3] and [4]). In this work we wanted to focus as much as possible on compute and communication performance and therefore we defined the test cases in such a way that I/O operations are minimized (i.e. only write final result data).

Another key factor for scalability is related to the solution of the systems of equations, both for the mesh motion and for the individual fluid and solid computations. As OpenFOAM uses iterative solvers the memory-bound nature of the code and process communication become a determining factor for the overall performance (see [5] and [6]). Profiling and scaling results with respect to these potential bottlenecks are discussed in Sections 4 and 5.

3. Solution algorithm and profiling setup

3.1. Implementation

Our implementation of an FSI-solver is derived from two sources, namely a coupled solver for fluids and solids of the OpenFOAM Extend Project [L2] (*icoFsiElasticNonLinULSolidFoam*) and a coupled thermo-dynamic solver of OpenFOAM 2.2.1 (*chtMultiRegionFoam*).

In terms of OpenFOAM two computational regions are defined, one for the fluid domain and one for the solid domain. Each domain has its own independent mesh, where the fluid mesh is defined as a dynamic mesh allowing transient mesh deformations. The time step loop contains an inner loop for the strong-coupling formulation which has the following structure.

1. Set interface displacement of fluid mesh (result of CSM computation). This step requires interpolation of displacement data from the solid domain to the fluid domain. For the case that all cells adjacent to the coupling interface are assigned to one process OpenFOAM's `patchToPatchInterpolation` is used (see Section 4.2). Otherwise `AMIPatchToPatchInterpolation` is used (see Section 4.3).
2. Compute mesh deformation of fluid mesh using one of OpenFOAM's mesh motion solvers.
3. Solve fluid fields using a pressure implicit scheme with splitting of operators (PISO) scheme.
4. Set interface pressure respectively forces for solid mesh (result of CFD computation). This step requires interpolation of pressure data from the fluid domain to the solid domain. Independent of the assignment of interface cells `AMIPatchToPatchInterpolation` is used.
5. Solve solid fields using a homogenous linear elastic material model.

Aitken relaxation is used for setting the interface displacement within the inner loop (see [7]). The termination criterion for the inner loop is a predefined relative residual for the interface displacement. Figure 1 shows the solution sequence of the strongly coupled solver, where $\boldsymbol{d}_i^{\Gamma,k_i^j}$ and $\boldsymbol{f}_i^{\Gamma,k_i^j}$ are the displacement respectively pressure (force) data on the coupling interface Γ for time step i and inner iteration k_i^j.

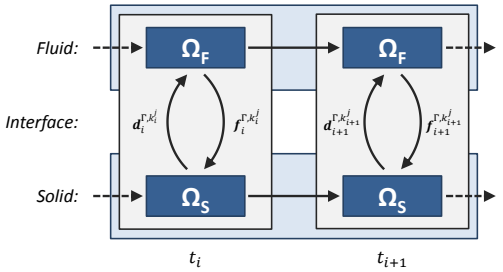


Figure 1: Solution sequence of strongly coupled fluid-structure solver.

Compared to a non-coupled simulation the following additional calculation steps are required.

- Interpolation of cell-centred displacement data to grid point locations within the solid mesh.
- Interpolation of grid point displacement from solid interface to grid point displacement of fluid interface.
- The relaxation scheme requires the computation of two inner products and three global sums over the interface.
- Interpolation of surface pressure from fluid interface to solid interface.
- Optional: For user information the global maximum displacement and the total pressure on the interface are computed.

The implementation utilizes two improved features of OpenFOAM 2.2: (1) Arbitrary mesh interface (AMI) for interpolating data between meshes on the coupling interface. (2) Motion solver for deforming the fluid mesh using one of the predefined motion solvers.

3.2. Installation notes

The profiling and scalability results reported in this report are based on OpenFOAM versions 2.2.1 and 2.3.0. All computations using the 3D beam model (Section 4) were done with standard debug builds of OpenFOAM on CURIE owned by GENCI and operated by CEA. The underlying MPI stack is Bullxmpi 1.1.16, which is a derivate of OpenMPI [L3]. Computations using the aircraft model (Section 5) were either done on Anselm at VSB – TU Ostrava or also on CURIE.

Due to the complexity of the source code and in some aspects unconventional programming style of OpenFOAM, we were first facing problems with profiling the application with TAU 2.21.2 [L4] or SCALASCA 1.4.3 [L5]. The main obstacle for source code instrumentation with TAU or SCALASCA is OpenFOAM's paradigm that commonly used code-snippets (at the level of C++ code blocks) are placed in separate source files which are directly included using preprocessor directives wherever needed. This seems to be something TAU's and SCALASCA's C and C++ parsers cannot cope with. Unfortunately also other attempts, using TAU's alternative features like compiler-based instrumentation and library interposition, did not yield satisfactory results.

For the above reasons we decided to use HPCToolkit 5.3.2 [L6], a suite of tools for performance measurement and analysis which is based on statistical sampling of timers and hardware performance counters. In addition to working out-of-the box in conjunction with OpenFOAM's peculiarities, HPCToolkit comes with a very good documentation as well as general guidelines for efficiently profiling parallel applications. Program instrumentation is done by using the hpcrun launch script which inserts profiling code via LD_PRELOAD. To have performance information attributed to loops and source code lines (and not only on a function-level), debugging information needs to be included into the binary. Apart from that, no additional changes to the source code or to the build process are required. The HPCToolkit manual recommends the use of a fully optimized debug build for profiling. However, we made the experience that function inlining obscures the actual code paths and can make a detailed interpretation of the generated profiles very hard. This is particularly the case for applications like OpenFOAM, having complex calling hierarchies and typically quite deep call paths.

4. Results: 3D beam model

In this section we document scalability results for our implemented FSI solver *icoFsiFoam* using a simplified "academic" test case, which we refer to as the 3D beam case. After a detailed profiling analysis, we present an optimization to OpenFOAM's core inter-process communication library (`Pstream` class), which significantly improves absolute runtimes and scalability of *icoFsiFoam*. Because the `Pstream` class is a fundamental concept of OpenFOAM's software architecture a broad range of operations benefit from this improved implementation.

One of our key findings is that perfomance and scalability of *icoFsiFoam* highly depend on the selected method for updating the fluid mesh. Mesh update is done by solving cell-centred Laplacian for the motion displacement (`displacementLaplacianFvMotionSolver` class). From the different variants that OpenFOAM offers, we used the uniform diffusivity model (as an example of a computationally simple method) and the quadratic inverse distance diffusivity model (as an example of a computationally complex method).

4.1. Mesh and mesh geometry

The 3D beam case is an idealized, easily scalable model of a box-shaped beam bending in a wind-tunnel. Despite its low geometrical complexity, from a software-architectural point of view this model already covers all aspects essential for demonstrating and testing the features and performance characteristics of the implemented FSI-solver. The two domains for fluid and solid are realized as separate, independent meshes. In particular, the vertices of the fluid and the solid mesh at the coupling interface do not necessarily have to be coincident.

The geometrical configuration of the 3D beam case can be seen in Figure 2. For meshing, we used the standard OpenFOAM utility *blockMesh* in combination with the tools *topoSet* and *splitMeshRegions* in order to split the generated purely hexahedral mesh into two separate and independent meshes for the fluid and solid domain respectively. As commonly used, we introduced cell grading for a higher mesh resolution near the coupling interface (see Figure 3). Moreover, from a typical industrial application we expect that the computational effort for the CFD computation is significantly higher than for the CSM computation. This expectation is well reflected in the 3D beam example. With respect to number of cells, the fluid mesh is approximately 50 times larger than the solid mesh.

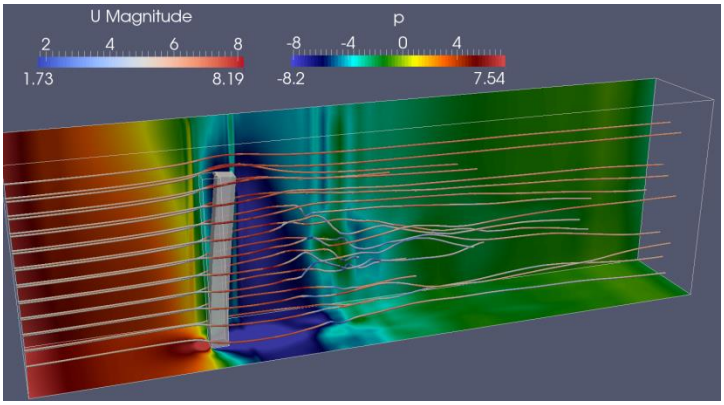


Figure 2: 3D beam geometry. A solid beam consisting of some homogeneous elastic material is bending in a wind tunnel. The ratio between the number of cells in the solid and fluid mesh is approximately 1:50.

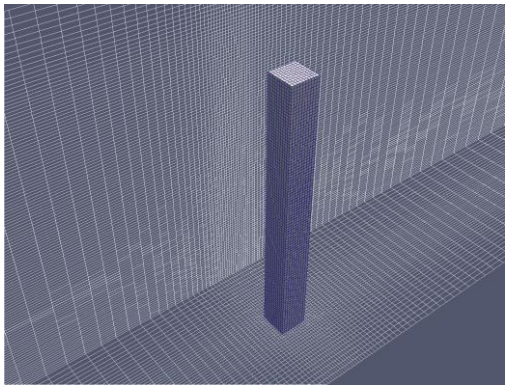


Figure 3: Cell grading in the 3D beam case (1 million cells). The spatial mesh resolution is highest near the coupling interface.

Due to the relative small problem size, the CSM computation does not scale as far as the CFD computation and best overall-performance is achieved when only a certain subset of all MPI processes (e.g. every 10th process) takes part in the CSM computation. For this reason, the solid mesh is decomposed into a smaller number of subdomains than the fluid mesh.

For the actual FSI computation, we initialize the flow field using a similar approach as in OpenFOAM's wing motion tutorial (*wingMotion2D_pimpleDyMFoam*). The flow field of the FSI case is initialized in two steps. First a converged solution to the (uncoupled) CFD problem on a coarser grid is derived using the OpenFOAM standard solver *icoFoam*. After that, the obtained flow field is mapped to the refined grid of the FSI case using OpenFOAM's *mapFields* utility.

4.2. Single-processor coupling scenario

If sophisticated conservative interpolation schemes have to be used or when coupling interfaces are defined by very complex geometries, data transfer between different meshes and/or solvers might require algorithmically complex implementations. In these cases it might be advantageous to do mesh decomposition in such a way that all cells adjacent to the coupling interface are assigned to one dedicated process.

In the course of our work we investigated how viable such an approach is in the case of FSI simulations. For our specific application domain, aircraft design, we estimate that the fraction of cells adjacent to the coupling interface is typically in the order of one per million. Hence it can be assumed that the "single-processor coupling" approach is mainly practicable for simulations with less than one-thousand cores only. In the context of OpenFOAM's domain decomposition strategy, it turns out that keeping the coupling interface on one processor has also subtle consequences for the quality of mesh decomposition.

4.2.1. Mesh decomposition

In Table 1 we compare two different mesh decomposition methods provided by OpenFOAM for partitioning the fluid mesh of the 3D beam example (meshed with 3.6 million cells) into 128 subdomains while keeping the coupling interface on one single processor.

The first approach, called simple decomposition, is based on a uniform space partitioning of the mesh's bounding box. It does not account for the finer mesh resolution near the coupling interface and therefore yields a higher imbalance with respect to the number of cells per subdomain compared to the case when cell grading is turned off. However, we intentionally go for graded meshes as we consider them more relevant for practice. The second decomposition method is based on the SCOTCH graph partitioning library.

Decomposition method	*Cell grading*	*Coupling interface* owner	*Maximum Cells per processor*	*Load imbalance*
Simple	off	-1 (automatic)	31 299 (Proc. 122)	14%
Simple	on	0 (fixed)	43 805 (Proc. 0)	59%
Simple	on	-1 (automatic)	43 805 (Proc. 0)	59%
SCOTCH	on	0 (fixed)	44 092 (Proc. 0)	61%
SCOTCH	on	-1 (automatic)	43 590 (Proc. 107)	59%

Table 1: Decomposition qualities for fluid mesh (3.6 million cells and 128 domains).
Load imbalance is measured by relating the maximum number of cells per processor to the average (≈ 28000).

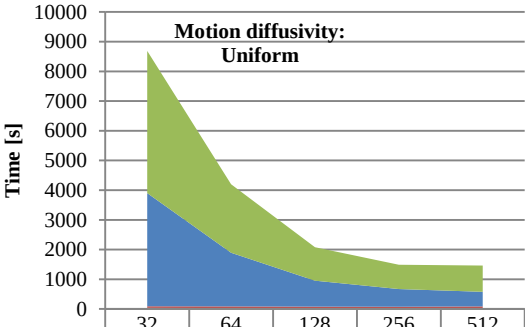
Independent of the selected decomposition method (simple or SCOTCH without weighting), we see that for meshes with cell grading the maximum number of cells per domain exceeds the average by about 60%. This obvious load imbalance is due to the fact that currently in OpenFOAM's decomposition configuration (*decomposeParDict*) the directive "singleProcessorFaceSets" is respected only in a post-processing step and not during the actual decomposition process. In other words, decomposition is first done without respecting "singleProcessorFaceSets" and then cells adjacent to the coupling interface are "stolen" from respective subdomains and added to one single subdomain.

A better load balancing can be achieved by specifying a weighting for the decomposition between processors for SCOTCH or by decomposing the mesh manually, i.e. by using the "manual" method. In order to minimize the influence of load balancing parameterization to scalability analysis, all results reported below are exclusively based on simple decomposition.

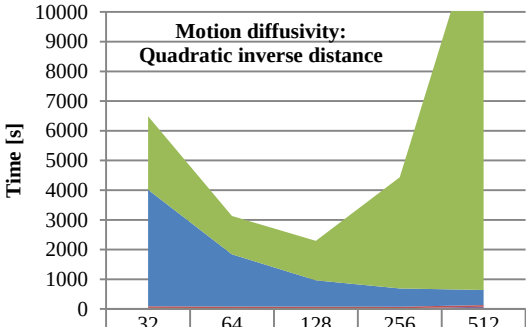
4.2.2. Scalability results

For studying the "single-processor coupling" scenario, we use a mesh of the 3D beam case with 3.6 million cells. In a strong scalability experiment, we measure the times of the FSI simulation for 10 time steps, varying the total number of MPI processes. Because of the comparatively small number of solid cells, the number of processes for the CSM computation is kept fixed at 8.

The total runtimes of the different FSI operations are compared in Figure 4. We can see that in the case of uniform diffusivity, scalability is near-to-optimal up to 128 processes and still acceptable for 256 processes. A totally different pitcture can be seen in the case of quadratic inverse distance diffusivity. Here the runtimes for mesh update increase significantly when using more than 128 processes. The reason for this behavior is discussed in detail in Section 4.4.



	32	64	128	256	512
Mesh	4787	2306	1126	818	884
Fluid	3811	1807	875	596	490
Solid	52	56	53	53	55
Other	38	30	24	21	32



	32	64	128	256	512
Mesh	2474	1291	1329	3744	12937
Fluid	3925	1758	887	615	509
Solid	52	52	54	54	55
Other	38	29	24	21	76

Figure 4: Total runtime of FSI operations for uniform (left) and quadratic inverse distance diffusivity (right).

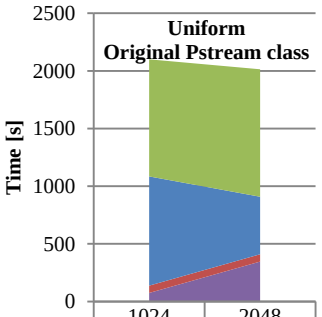
4.3. Multiple-processor coupling scenario

The interpolation and data transfer between meshes in icoFsiFoam relies on OpenFOAM's arbitrary mesh interface (AMI). This makes it possible to distribute the coupling interface for both regions among multiple MPI processes, i.e. using an m to n communication pattern. In this section we investigate the strong scalability behaviour of *icoFsiFoam* in this scenario and again compare the two variants for mesh update, uniform diffusivity and quadratic inverse distance diffusivity.

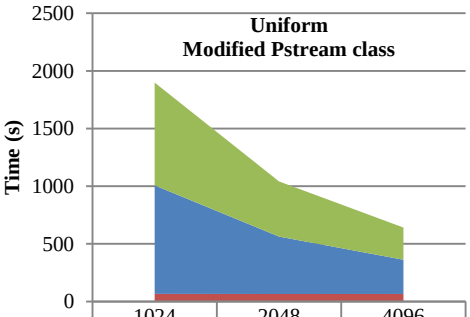
4.3.1. Scalability results

For studying the "distributed coupling scenario" we use a mesh of the 3D beam case with 61 million cells. Again we perform a strong scalability experiment (1024 up to 4096 processes) for 5 simulation time steps. The number of processes for the CSM (Solid) computation is left constant with 32 processes.

For uniform diffusivity Figure 5 compares the scalability behaviour of the different FSI operations when using OpenFOAM's original `Pstream` class or a modified variant which will be discussed in detail in Section 4.5. Using the original `Pstream` class, except for the CFD (Fluid) computation, we observe no scaling from 1024 to 2048 processes. For 4096 processes we do not have measurements available because the configured wallclock limit of 90 minutes was reached before the first of the 5 simulation time steps had finished. In contrast to that, on the right side of Figure 5 we see a good overall scalability up to 4096 processes when using our modified version of the `Pstream` class.

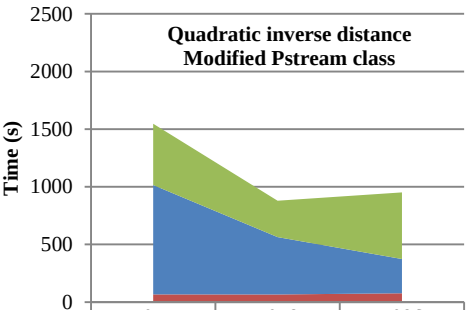


	1024	2048
Mesh	1016	1105
Fluid	947	499
Solid	64	63
Other	74	346



	1024	2048	4096
Mesh	892	478	279
Fluid	941	498	297
Solid	63	63	64
Other	3	2	2

Figure 5: Total runtime for FSI operations for uniform diffusivity. Scalability is considerably improved when using a modified version of OpenFOAM's Pstream library (right) instead of the original one (left).



	1024	2048	4096
Mesh	531	315	578
Fluid	948	498	297*
Solid	63	64	72
Other	4	2	4

(*) See text below.

Figure 6: Total runtime for FSI operations for quadratic inverse distance diffusivity using the modified Pstream class.

When using quadratic inverse distance diffusivity in combination with the original `Pstream` library, things get even worse. In this case already a simulation with 1024 processes is infeasible, as the wall clock time for the first of the 5 simulation time steps is roughly 1 hour (3530s). In contrast to that, when using the modified `Pstream` class the simulation with 1024 processes finishes all 5 time steps in less than half an hour, as shown in Figure 6. Moreover, good scalability in the range from 1024 to 2048 processes can be observed.

Interestingly, measured wall clock time in Figure 6 for fluid computation with 4096 processes was 863 seconds. Due to the implementational reasons this time has to be very close to the corresponding time given in Figure 5 (the same CFD solver is used for the same model). Therefore, we consider it necessary to verify the 863 seconds in order to rule out a singular errounous measurement. Unfortunately, we have not had the opportunity for a second measurement, and therefore, we assume time for fluid computation to be 297 seconds, based on data shown in Figure 5.

4.4. Performance analysis

Summing up the scalability results for both of the above scenarios (single processor coupling and distributed coupling), we see that mesh update with quadratic inverse distance motion diffusion has a very negative impact to scalability (compared to uniform motion diffusion). This fact can most clearly be seen in Figure 4 and renders mesh update of the fluid mesh as the most critical and performance-limiting operation.

For mesh update, we use OpenFOAM's so-called *displacementLaplacian* motion solver. As its name suggests, this motion solver propagates the displacement given at the coupling interface (as a result of the CSM computation) to the interior of the fluid domain by solving a Laplace equation with according boundary conditions. When using quadratic inverse distance motion diffusion, the diffusion coefficient of the Laplace equation at a given point P within the fluid domain is indirectly proportional to the squared Euclidean distance of P to the coupling interface. As a consequence, cells near the coupling interface move more freely than cells farther away. We consider this approach to be more relevant and applicable in conjunction with complex mesh geometries (such as aircrafts in high-lift configurations) than for example uniform motion diffusion (where the diffusion coefficient is assumed to be constant).

Analysis of the generated profiles with HPCToolkit reveals that it is in fact the distance calculation which is the most time-consuming operation. The coefficients of the motion Laplace equation are updated in every inner iteration of the strong coupling scheme. For this purpose a moving front algorithm is used, which starts at the boundary layer of the coupling interface and visits all cells of the fluid mesh exactly once. Data exchange needs to take place for every iteration of the moving front algorithm. It turns out that exactly this data exchange is a crucial bottleneck for scalability.

4.5. A modified implementation for OpenFOAM's Pstream class

Within OpenFOAM, inter-process communication is implemented in such a way that every MPI process writes output data into dedicated binary buffers (`PsteamBuffers` class). The actual data transfer is delayed to a collective synchronization point, when all MPI processes call the method `PstreamBuffers::finishedSends()`. After that, every MPI process reads back the input data from binary buffers.

The method `PstreamBuffers::finishedSends()` delegates the data transfer to the method `Pstream::exchange()`. Surprisingly, the hotspot within `Pstream::exchange()` is not the transfer of the actual messages, but the transfer of the message sizes, which is done by the function `combineReduce()` (see Figure 7). Two things are worth mentioning.

- More data than necessary is transferred: `combineReduce()` replicates the complete matrix of message sizes on all processors, although in general only one row of this matrix is really relevant to a given processor.
- The implementation is not optimal: `combineReduce()` emulates a collective operation by using MPI point-to-point communication routines rather than using appropriate collective communication MPI routines directly.

In order to address these two observations, we changed the implementation of `Pstream::exchange()` in the following way. For the problem at hand, namely the transfer of message sizes, the most natural approach is to use the collective communication routine `MPI_Alltoall()`. We added a simple wrapper method for `MPI_Alltoall()` to the class `Upstream` and replaced the function call `combineReduce()` in the method `Pstream::exchange()` by a call to the new method `Upstream::alltoall()`. This way every MPI process receives just the data needed.

Figure 7 shows how this optimization affects the performance of *icoFsiFoam*, running the 3D beam case with 512 processes in the single-processor coupling scenario with a 3.6 million cells mesh using quadratic inverse distance motion diffusion. We see that the runtime contribution of `Pstream::exchange()` dropped from 60% to 9.3%. The absolute runtime, as reported by HPCToolkit's PAPI_TOT_CYC event for measuring total cycles, dropped from 6.14×10^{14} CPU cycles to 7.41×10^{12} CPU cycles, which corresponds to a speedup factor for this function of more than 80.

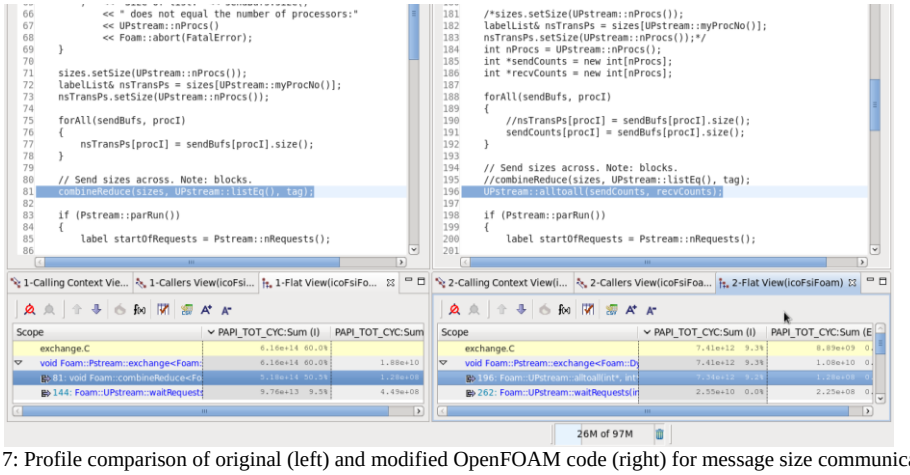


Figure 7: Profile comparison of original (left) and modified OpenFOAM code (right) for message size communication. The total runtime contribution of Pstream::exchange() dropped from 60% to 9.3% (3.6M cells, 512 processes).

The modified implementation has also a positive effect when using uniform motion diffusion. Looking at Figure 5 we see that the strong scalability of the 61 million cells case from 1024 to 2048 processes is greatly improved (see runtimes for "Mesh" and "Other").

5. Results: Aircraft model

In the context of industrial applications it is very common that numerical models, i.e. meshes, are created using one software system, as e.g. ICEM CFD or HyperMesh, and computation is carried out subsequently by another package. In fact there is no open source package which could create meshes on top of geometries exported from commonly used CAD/CAM packages such as CATIA, ProEngineer, or Invertor etc. For this reason we consider for the industrial example also these two tasks, namely mesh generation and solution computation.

5.1. Existing mesh from external source

First an existing mesh of the fluid domain surrounding the aircraft obtained from publicly available source was used. This model was available in CFD General Notation System (CGNS) format, which consists of a collection of conventions and provides a general standard for storage and retrieval of CFD analysis data. Since OpenFOAM uses a different system for storing mesh data, conversion between CGNS and OpenFOAM format had to be done. A conversion tool was developed within the OpenFOAM Extend Project. However this tool is outdated, it was developed for OpenFOAM version 1.5, and we were not able to successfully convert the mesh using this tool.

In a second step we used the following procedure to convert the mesh from CGNS and OpenFOAM format.

- CGNS format was converted into ANSYS Fluent format using commercially available package ICEM CFD.
- Fluent file format was converted into OpenFOAM format by *fluentMeshToFoam* routine available within OpenFOAM distribution.

Three meshes of fluid domain consisting of approximately 6, 22, and 50 million of cells were converted, mesh quality check was performed and first test runs were performed to establish whether those meshes could be used for further production runs. Quality check revealed that all meshes consist of boundary layers in areas close to aircraft body, where aspect ratios of cells exceed the value of 2000. Test runs confirmed that such meshes cannot directly be used with OpenFOAM because its solvers are quite sensitive to mesh quality. Another issue with pre-existing mesh regards storing of huge data files, which in case of meshes consisting of hundreds of millions or even billions of cells, mean Terabytes of data which has to be uploaded, stored and loaded by solver.

5.2. Mesh generated directly by OpenFOAM

As a consequence of the unsuccessful attempts of using existing meshes of the fluid domain surrounding aircraft, we decided to use OpenFOAM's own mesh generation tool. OpenFOAM offers several tools for this task.

- For simple geometries *blockMesh* can be used, which is a multi-block mesh generator that creates meshes of hexahedra based on a text configuration file.
- For complex geometries there is *snappyHexMesh* that uses triangulated surface meshes in STL format as an input for volume mesh generation. The *snappyHexMesh* utility is fully parallel and can generate meshes of hundreds of millions or even billions of cells. The parallel version of *snappyHexMesh* generates meshes in such a way that each process creates a subdomain of the global mesh, which is stored individually and can directly be used by the solver. For this reason the number of processes for mesh generation has to be same as for the solution part.
- To generate mesh by extruding cells from a patch of an existing mesh, or from a surface mesh *extrudeMesh* utility can be used.
- The *polyDualMesh* utility creates the dual of a polyMesh. It can for example be used to generate a "honeycomb" polyhedral mesh from a tetrahedral mesh.

In our case the *snappyHexMesh* utility was used to create meshes consisting of approximately 105, 235, and 845 million of cells for the fluid domain and approximately 5 million of cells for the solid domain. Figure 8 shows the geometry of aircraft in STL format which was used for mesh generation by *snappyHexMesh*.

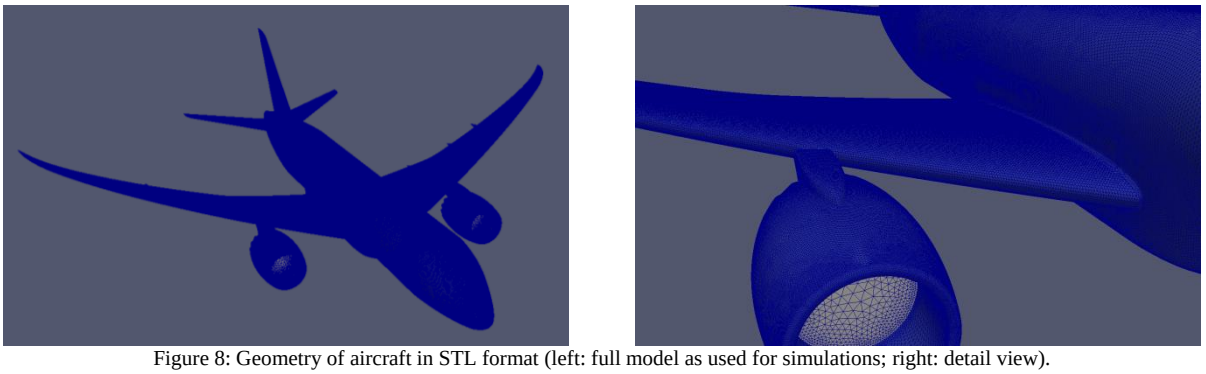


Figure 8: Geometry of aircraft in STL format (left: full model as used for simulations; right: detail view).

As already mentioned in Section 4.1 it is very common for industrial applications that mesh sizes of CFD and CSM differ by several orders of magnitude. This was the reason why we also created such different discretizations for CFD and CSM.

5.3. Simulation setup

For fluid-structure interaction (FSI) simulations the following four different subtasks have to be performed. (1) CSM solution, (2) data transfer from CSM to CFD including CFD mesh update, (3) CFD solution, and (4) data transfer from CFD to CSM.

All those subtasks together define the overall performance and efficiency of an FSI solver. In this section scalability tests of CFD and CSM simulations together with 0th order FSI simulation, where the displacement of the mesh is prescribed as a boundary conditions, are presented. For CFD *icoFoam*, a solver for incompressible laminar flow, was used. For CSM *solidDisplacementFoam* was used, which models a linear elastic case with small displacements. For solving the case of prescribed displacement at the coupling interface we used *pimpleDMyFoam*. Figure 9 shows exemplary results of the CFD computation.

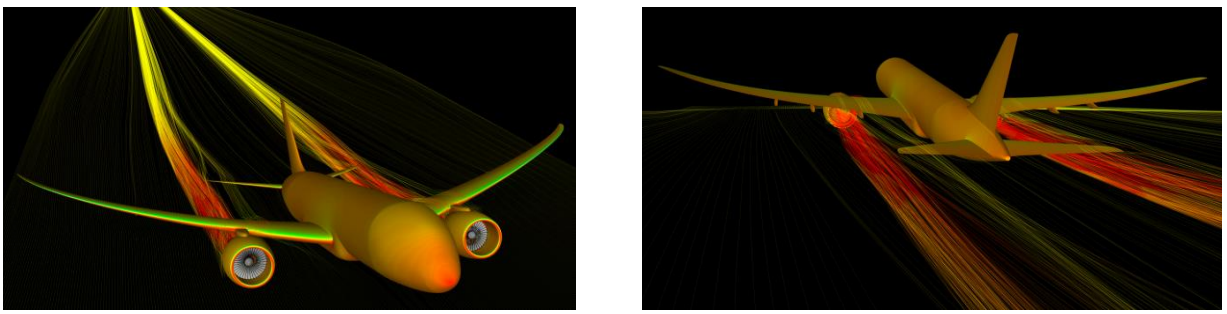


Figure 9: Streamlines around aircraft computed by *icoFoam*.

All numerical experiments were carried out on two systems. For CSM calculations (mesh consisting approximately of 5 million of cells) and CFD calculations up to 1024 cores (mesh consisting of approximately 105 million of cells) HPC system Anselm at VSB – TU Ostrava was used. For CFD calculations up to 2048 cores (two meshes consisting of approximately 237 and 845 million of cells respectively) and FSI calculations up to 2048 cores (mesh consisting of approximately 180 million of cells) the Tier-0 system CURIE at CEA was used. In all cases the time necessary for meshing of the domains and the computational time were of interest and are reported in this paper.

5.4. CFD calculations

First CFD simulations using the numerical model consisting of approximately 105 million of cells were performed. The mesh was generated in parallel by *snappyHexMesh* utility and CFD calculations for 1000 time steps were performed by *icoFoam* solver. Table 2 shows the runtimes needed for both operations for 128, 256, 512, and 1024 cores respectively. We were unable to use less than 128 cores due to memory demand for creating the 105 million cells mesh.

Cores	*Meshing [s]*	*CFD [s]*	*Total [s]*	*Cost [core h]*
128	5 180	12 420	17 600	626
256	3 890	6 300	10 190	725
512	3 839	2 880	6 719	956
1 024	4 086	1 404	5 490	1 562

Table 2: Times needed for mesh generation and computation of 105 million cells model.

After analysing the results of scalability tests on the model with 105 million of cells we observed that scalability of parallel meshing is very poor (see column 2 of Table 2). To test whether this very poor scalability of parallel meshing tool *snappyHexmesh* is due to small number of cells per core, another numerical experiment with a mesh of approximately 235 million of cells were carried out, this time on Tier-0 system CURIE. Results of those tests are presented in Table 3.

Cores	*Meshing [s]*	*CFD [s]*	*Total [s]*	*Cost [core h]*
256	7 646	4 615	12 261	871
512	5 396	2 378	7 774	1 105
1 024	6 918	1 428	8 346	2 373
2 048	17 693	5 497	23 190	13 192

Table 3: Times needed for mesh generation and computation of 235 million cells model.

Due to memory demands we were unable to use less than 256 cores. Table 3 shows that CFD simulation scales up to 1024 cores, whereas mesh generation scales only up to 512 cores and even than it is not very efficient. It seems that some limits of the mesh generation tool *snappyHexMesh* were reached in term of meshes generated in reasonable time and with reasonable costs.

To test the limits of the CFD solver last tests were performed. This time 1000 time step iterations on a mesh consisting of approximately 845 million of cells for 1024 and 2048 cores were performed. We were unable to use less than 1024 cores and after obtaining results we decided not to use more than 2048 cores to save computational resources. Results of those tests are show in Table 4.

Cores	*Meshing [s]*	*CFD [s]*	*Total [s]*	*Cost [core h]*
1 024	12 863	8 192	21 055	5 989
2 048	23 434	9 937	33 371	18 984

Table 4: Times needed for mesh generation and computation of 845 million cells model.

As can be seen from the results in Table 4 limits for CFD computation were reached as well. For the mesh consisting of 845 million of cells less than 1024 cores could not be used and using more than that did not decrease computational time.

5.5. CSM calculations

Since we did not have a real structural model of the aircraft we meshed the entire interior with 3D elements and solved a pseudo structural model instead. In fact, for real models most of the parts such as aircraft skin etc. would be simplified and modelled by shell elements and finite elements would be used for discretization rather than finite volumes as in the case of OpenFOAM. Because our main goal are scalability tests, this is an admissible approach to get an overview about the CSM solver for cases with millions of degrees of freedom. As we mentioned earlier the number of cells for the structural model is approximately 5 millions which means that we have to solve systems of equations with approximately 15 millions of unknowns. All CSM tests were run on the HPC system Anselm at VSB – TU Ostrava and results are shown in Table 5. In all cases 1000 time step iterations of *solidDisplacementFoam* were run.

Cores	*Meshing [s]*	*CSM [s]*	*Total [s]*	*Cost [core h]*
32	860	863	1 723	15
64	435	431	866	15
128	302	223	525	19
256	294	164	458	33
512	471	109	580	82
1 024	1 736	86	1 822	518

Table 5: Times needed for mesh generation and CSM computation of 5 million cells model.

5.6. Dynamic mesh calculation

To test runtime performance of CFD with mesh update the solver *pimpleDMyFoam* was selected and a numerical model consisting of approximately 185 million of cells was created. On this model 10 time steps with prescribed displacements of aircraft wing were run. Table 6 shows results obtained on Tier-0 system CURIE.

Cores	*Meshing [s]*	*CSM [s]*	*Total [s]*	*Cost [core h]*
128	2 868	3 842	6 710	239
256	2 635	2 021	4 656	331
512	4 643	875	5 518	784
1 024	8 927	485	9 412	2 677
2 048	28 990	409	29 399	16 724

Table 6: Times needed for mesh generation and CSM computation of 185 million cells model.

6. Conclusion and outlook

Based on OpenFOAM a partitioned, strongly coupled solver for transient FSI simulations with independent meshes for the fluid and solid domains has been implemented. For several variants of solvers and models runtime and scalability analyses have been performed. These results indicate that updating the CFD mesh is the most significant operation with respect to strong scalability. On the other hand data exchange and interpolation on the coupling interface – if it is distributed among the processes in a balanced way – has a minor effect on the total runtime.

By changing the implementation of OpenFOAM's inter-process communication class from point-to-point MPI communication routines to collective MPI communication routines scaling characteristics could be enhanced

considerably. The most important consequence of this modification is a substantial improvement for mesh update. For a model with 61 million cells and uniform diffusivity for mesh update good speedups can be achieved up to 4096 processes. If for the same model quadratic inverse distance diffusivity is used, the scalability limit for mesh update lies between 2048 and 4096 processes. The changes will be reported to the developers of the official OpenFOAM release in order to discuss how the implementation of inter-process communication can be improved in future releases. As a representative example for an industrial use case, a test case of a full aircraft model has been created and scaling analyses of the individual sub-computations have been performed.

Within this work changes of OpenFOAM's inter-process communication were exclusively guided by profiling results with respect to FSI computations. For future work we see very concrete potential for improving the implementation of inter-process communication further, having a significant positive impact on system-wide performance.

Acknowledgments

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763.

References

[1] PRACE – The Scientific Case for HPC in Europe 2012-2020. From Petascale to Exascale. PRACE publication. 2012.

[2] PRACE-2IP Deliverable 9.1.1. Support for Industrial Applications Year 1. 2012.

[3] Michael Moylesa, Peter Nash, Ivan Girotto. Performance Analysis of Fluid-Structure Interactions using OpenFOAM. PRACE report.

[4] Bjørn Lindi. I/O-profiling with Darshan. PRACE report.

[5] Massimiliano Culpo. Current Bottlenecks in the Scalability of OpenFOAM on Massively Parallel Clusters. PRACE report.

[6] Orlando Rivera, Karl Fürlinger, Dieter Kranzlmüller. Investigating the Scalability of OpenFOAM for the Solution of Transport Equations and Large Eddy Simulations. ICA3PP'11 Proceedings of the 11th international conference on Algorithms and architectures for parallel processing – Volume Part II, LNCS 7017, pp 121-130. 2011.

[7] Ulrich Küttler, Wolfgang A. Wall. Fixed-point fluid–structure interaction solvers with dynamic relaxation. Comput Mech 43, pp 61–72. 2008.

[L1] http://www.openfoam.com

[L2] http://www.extend-project.de

[L3] http://www.open-mpi.org

[L4] http://www.cs.uoregon.edu/research/tau/home.php

[L5] http://www.scalasca.org

[L6] http://hpctoolkit.org