CUPyDO - An integrated Python environment for coupled fluid-structure simulations

D. Thomas[a,*], M.L. Cerquaglia[a], R. Boman[a], T.D. Economon[b], J.J. Alonso[b], G. Dimitriadis[a], V.E. Terrapon[a]

[a] *Department of Aerospace and Mechanical Engineering*
University of Liège
Liège, 4000, Belgium
[b] *Department of Aeronautics and Astronautics*
Stanford University
Stanford, CA, 94305, USA

Abstract

CUPyDO, a fluid-structure interaction (FSI) tool that couples existing independent fluid and solid solvers into a single synchronization and communication framework based on the Python language is presented. Each coupled solver has to be wrapped in a Python layer in order to embed their functionalities (usually written in a compiled language) into a Python object, that is called and used by the coupler. Thus a staggered strong coupling can be achieved for time-dependent FSI problems such as aeroelastic flutter, vortex-induced vibrations (VIV) or conjugate heat transfer (CHT). The synchronization between the solvers is performed with the predictive block-Gauss-Seidel algorithm with dynamic under-relaxation. The tool is capable of treating non-matching meshes between the fluid and structure domains and is optimized to work in parallel using Message Passing Interface (MPI). The implementation of CUPyDO is described and its capabilities are demonstrated on typical validation cases. The open-source code SU2 is used to solve the fluid equations while the solid equations are solved either by a simple rigid body integrator or by in-house

*Corresponding author at: University of Liège, Department of Aerospace and Mechanical Engineering, Allée de la Découverte, 9, 4000, Liège, Belgium.

Email addresses: dthomas@ulg.ac.be (D. Thomas), marcolucio.cerquaglia@ulg.ac.be (M.L. Cerquaglia), r.boman@ulg.ac.be (R. Boman), economon@stanford.edu (T.D. Economon), jjalonso@stanford.edu (J.J. Alonso), gdimitriadis@ulg.ac.be (G. Dimitriadis), vincent.terrapon@ulg.ac.be (V.E. Terrapon)

linear/nonlinear Finite Element codes (GetDP/Metafor). First, the modularity of the coupling as well as its ease of use is highlighted and then the accuracy of the results is demonstrated.
Keywords: Coupling Environment, Python Wrapper, Fluid-Structure Interaction, Partitioned Coupling, Computational Aeroelasticity, CUPyDO

1. INTRODUCTION

The study of fluid-structure interaction (FSI) problems is an important center of interest for researchers and engineers in a significant range of applications in different fields such as aerospace [1, 2, 3, 4], biomedical/biological [5, 6, 7] and civil engineering [8], to cite a few. The drastic evolution of the available computational power allows the use of advanced and complex models for each type of physics involved. These models are usually based on high-fidelity three-dimensional frameworks in order to quantify phenomena that low-order models are not able to reproduce. Such high-fidelity computation of FSI problems is usually based on one of two possible strategies: the monolithic or the partitioned approach [9, 10]. In the monolithic approach, both the structural and fluid problems are solved by a single solver and within the same mathematical framework, where the interfacial conditions are implicit to the procedure. This requires significant coding efforts and the implementation is usually designed to accommodate a particular case of interest, often leading to a lack of generality. On the other hand, the partitioned approach couples two different specialized existing codes that are used to solve separated sub-systems. This requires an efficient communication and synchronization interface but allows the intrinsic features of the individual solvers to be exploited. If the implementation of the coupling interface is flexible enough and compatible with many different solvers, a wide range of coupled physics and related numerical methods can be considered for many different applications. In a partitioned approach, we distinguish two coupling schemes [11]: weak and strong coupling. The weak coupling scheme requires only one solution of the sub-systems per time step,

which is the most efficient choice in terms of computational time but could lead to a time-lagged solution. Also, numerical instabilities may appear for problems that involve strong added-mass effects [11], i.e. when the fluid and solid densities are similar and/or the solid is very flexible. If not properly identified, those numerical instabilities might be erroneously treated as physical instabilities such as wing flutter. The strong coupling scheme requires solving several times the sub-systems per time step in order to guarantee the convergence at the interface and gives the same results as the monolithic approach but with a larger simulation time compared to weak coupling. Strong added-mass effects are usually better handled but can still lead to a non-convergent iterative process.

An integrated framework to strongly couple two heterogeneous existing fluid and solid solvers is presented in this paper. It is called CUPyDO and consists in a Python coupling environment designed to interact with the two solvers through an API Python wrapped layer. Data exchange and synchronization are thus implemented in a very intuitive and flexible way in Python, whereas the computationally intensive routines within each solver are kept in their native languages (C, C++, etc.). Because the coupled solvers are reduced to black-box tools by their wrappers, minimal effort is required to achieve compatibility with the coupler ensuring significant flexibility in solver choice. Other workers in the field have developed coupling schemes between different solvers; Bungartz *et al.* [12] give a list of some current implementations. One of the most standard tool is the commercial software MpCCI [13], that proposes ready-to-use adapters to many commercial solvers. The client-server architecture of MpCCI leads to coupling frameworks of potentially high complexity but limits thus the parallel scalability of large systems by serializing important coupling tasks. ADVENTURE_Coupler [14] is an open-source coupling tool for large-scale problems that is also based on a client-server architecture. The coupling compatibility is performed by intrusive coupler-specific routines that have to be added in the solver core code. OpenPALM [15] is another open-source software for massively parallel coupled applications. Functionalities from the coupler have to be explicitly added in the coupled core codes and the communications between the coupled

systems rely on the MPI protocole. The preCICE open-source coupling framework [12] is designed to maintain an efficient parallel scalability by avoiding the use of a server instance. The communication between the coupled codes are still based on MPI or TCP/IP protocoles and intrusive specific routines need to be introduced in the coupled core codes from the coupling library.

In this paper, an open-source coupling environment, based on the Python wrapping methodology, is proposed as an alternative integrated implementation that does not rely on MPI or TCP/IP protocols for communications between the coupled modules. In CUPyDO, emphasis is given to the parallel implementation and to the modularity of the coupling mechanism with no explicit coupler-specific routines to be introduced into the coupled codes, thus minimizing intrusive code modification. User-friendliness is also ensured by providing ready-to-use coupling functionalities, such as interpolation methods for non-matching grids or iterative algorithms, but also by limiting the amount of actions required by the user to set and launch a coupled simulation of low complexity. Although the coupling mechanism of CUPyDO is demonstrated here on fluid-structure interaction problems, such as flutter, vortex-induced vibrations and steady conjugate heat transfer, the infrastructure is designed in a general way to allow an easy extension to many other multi-physics problems.

The paper is organized as follows. Section 2 is dedicated to describing the governing equations of the coupled problem. Fluid and solid equations as well as the interface conditions are introduced. The different solvers used to solve FSI problems are also presented. The implementation of CUPyDO is then detailed in Section 3. Particular attention is paid on the solvers interfacing mechanism, the way the communication with the solvers is performed and the parallel implementation. In Section 4 several FSI test cases are reproduced using different coupled solvers and results are presented in order to show the accuracy and the flexibility of the coupling tool. Finally, Section 5 summarizes the main concepts and results, and suggests further steps for future work.

2. GOVERNING EQUATIONS OF THE COUPLED PROBLEM

This section describes the governing equations and the numerical implementations used in this paper for both the fluid and solid parts of the coupled problem. The fluid equations, in an Arbitrary Lagrangian-Eulerian (ALE) formulation, are solved in a moving and conforming fluid domain Ω_{f} that shares a common boundary Γ with the solid domain Ω_{s}, in which the solid equations are solved in a Lagrangian formulation. In addition to intrinsic boundary conditions for each of the disciplines, coupling conditions on the displacements, the loads, the temperatures and the heat fluxes across the common boundary Γ are required to achieve a strong coupling scheme.

2.1. Fluid mechanics

The dynamic behavior of a compressible Newtonian fluid is predicted by solving the Navier-Stokes or Euler equations. The conservation of mass, momentum and energy in Ω_{f} can be expressed as [16]

$$\frac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot \boldsymbol{F}^{\mathrm{c}} - \nabla \cdot \boldsymbol{F}^{\mathrm{v}} = \boldsymbol{Q} \qquad \text{in } \Omega_{\mathrm{f}} \times [0, t] \ , \tag{1}$$

where the conservative variables are given by $\boldsymbol{U} = [\rho, \rho\mathbf{v}, \rho E]^{\mathrm{T}}$. The advective and diffusive fluxes are given by

$$\boldsymbol{F}^{\mathrm{c}} = \begin{bmatrix} \rho\mathbf{v} \\ \rho\mathbf{v} \otimes \mathbf{v} + p\mathbf{I} \\ \rho E\mathbf{v} + p\mathbf{v} \end{bmatrix} , \qquad \boldsymbol{F}^{\mathrm{v}} = \begin{bmatrix} \cdot \\ \boldsymbol{\tau} \\ \boldsymbol{\tau} \cdot \mathbf{v} + \mu^{*}_{\mathrm{tot}} c_p \nabla T \end{bmatrix} , \tag{2}$$

and $\boldsymbol{Q}$ is a source term. In these expressions ρ is the fluid density, $\mathbf{v}$ the velocity field, E the total energy per unit mass, p the static pressure, c_p the specific heat capacity at constant pressure, and T the temperature. For a Newtonian fluid, the viscous stress tensor is given by

$$\boldsymbol{\tau} = \mu_{\mathrm{tot}} \left(\nabla\mathbf{v} + \nabla\mathbf{v}^{\mathrm{T}} - \frac{2}{3}\mathbf{I}(\nabla \cdot \mathbf{v}) \right) , \tag{3}$$

and the system is closed with state equations and boundary conditions. Note that the Euler equations are recovered by discarding the viscous terms.

Viscous turbulent flows are modelled using the Reynolds-Averaged Navier-Stokes equations for which an additional eddy viscosity is calculated by suitable turbulence models such as the Shear Stress Transport k-ω [17] or the Spalart-Allmaras (SA) [18] model. The contribution of the turbulent viscosity to the viscous terms is given by

$$\begin{aligned} \mu_{\mathrm{tot}} &= \mu_{\mathrm{d}} + \mu_{\mathrm{t}} \,, \\ \mu^{*}_{\mathrm{tot}} &= \frac{\mu_{\mathrm{d}}}{\mathrm{Pr}_{\mathrm{d}}} + \frac{\mu_{\mathrm{t}}}{\mathrm{Pr}_{\mathrm{t}}} \,, \end{aligned} \tag{4}$$

where μ and Pr are the viscosity and Prandtl number, respectively. The subscript "d" stands for dynamic and expresses intrinsic properties of the fluid, whereas the subscript "t" expresses quantities coming from turbulence modelling. The contribution of the turbulent quantities are removed for laminar flow computations.

Equations (1)-(2) are written for a purely Eulerian formulation and do not account for any motion of the computational domain that may appear in the context of fluid-structure interaction problems. The Arbitrary Lagrangian-Eulerian formulation [19] takes into account the motion of Ω_{f} by adding its contribution to the advective fluxes in (2):

$$\boldsymbol{F}^{\mathrm{c}} = \begin{bmatrix} \rho(\mathbf{v} - \mathbf{v}_{\Omega}) \\ \rho\mathbf{v} \otimes (\mathbf{v} - \mathbf{v}_{\Omega}) + p\mathbf{I} \\ \rho E(\mathbf{v} - \mathbf{v}_{\Omega}) + p\mathbf{v} \end{bmatrix} , \tag{5}$$

where $\mathbf{v}_{\Omega}$ is the local velocity field of the moving computational domain. Purely Eulerian or purely Lagrangian formulations can be recovered by setting $\mathbf{v}_{\Omega} = 0$ or $\mathbf{v}_{\Omega} = \mathbf{v}$, respectively.

The open-source CFD code SU2 [16, 20, 21, 22, 23] is used to solve the fluid part of the coupled problem in ALE. The governing equations are spatially discretized using the Finite Volume Method on a dual-grid using a vertex-based approach so that the nodes of the primal grid represent the centers of the control volumes. Temporal discretization is achieved through a dual time-stepping [24] strategy, where each physical time step is transformed into a steady problem for which steady state acceleration techniques can be used. The grid motion

is considered as a part of the fluid problem. It is usually computed by one of three approaches: the spring analogy where the mesh edges are replaced by springs [25, 26], the solution of a Laplace equation [27] or the solution of a pseudo-elastic solid problem. In SU2, the grid deformation is computed at each time step by a steady pseudo-elastic solid problem where the mesh is casted as a deformable solid, and whose boundary conditions correspond to the motion of the wetted solid surface. The governing equation for the displacement of the mesh nodes $\mathbf{d}_\Omega$ is based on a finite element approach and given by:

$$\mathbf{K}_\Omega \mathbf{d}_\Omega = \mathbf{f}_\Omega \,, \tag{6}$$

where $\mathbf{K}_\Omega$ is a fictitious stiffness matrix and $\mathbf{f}_\Omega$ a fictitious force to enforce the boundary motion. A variable Young's modulus is defined in order to control the mesh quality in specific regions. Typically, high stiffness is required for cells with a large aspect ratio in wall boundary layers in order to minimize their deformation. Lower stiffness is used for larger cells in the farfield, where they can withstand larger deformations. The grid velocities are then computed using a consistent finite difference scheme from the positions of the mesh nodes stored at previous time steps. The Geometric Conservation Law (GCL) [28], that has to be satisfied for unsteady flows on moving meshes computed with the ALE formulation, is numerically implemented in SU2 as part of the dual time-stepping procedure [22, 29]. Although only the SU2 solver is used, another fluid solver based on the PFEM formulation [30, 31] has recently been coupled with CUPyDO but no application involving this coupling are treated in this paper.

2.2. Solid mechanics

The dynamic behavior of a deformable solid results from the balance between inertial, internal and external forces. The equilibrium equation in Ω_s is given by

$$\rho \frac{\partial^2 \mathbf{d}}{\partial t^2} - \nabla \cdot \boldsymbol{\sigma} = \mathbf{f} \qquad \text{in } \Omega_\mathrm{s} \times [0, t] \,, \tag{7}$$

where ρ, $\mathbf{d}$, $\boldsymbol{\sigma}$ and $\mathbf{f}$ are the solid density, the displacement vector, the Cauchy stress tensor and the body forces, respectively. The temperature field within a

solid is obtained by solving the linear heat equation in Ω_s:

$$\rho c_p \frac{\partial T}{\partial t} - \nabla \cdot (\lambda \nabla T) = Q_v \qquad \text{in } \Omega_\text{s} \times [0, t] \ , \tag{8}$$

where λ and Q_v are the thermal conductivity and a possible volume heat source, respectively. Equations (7) and (8) are coupled by considering the decomposition of the total deformation between a mechanical and a thermal part, the latter being dependent on the thermal field. Plastic deformations and friction forces during contacts may also produce heat sources in Equation (8).

The in-house nonlinear Finite Element code Metafor [32, 33, 34, 35, 36, 37] is one of the solvers used to compute the structural part of the coupled problem. The solver is designed to simulate large structural deformations in a Lagrangian formulation by expressing the principle of virtual work (PVW) on the deformed configuration. A particular feature of Metafor is also the large range of nonlinear material laws that can be used (elasticity, elasto-plasticity or visco-elasto-plasticity, etc.). For each time increment, the equations of motion are solved using a Newton-Raphson approach:

$$\begin{aligned} \mathbf{M}\ddot{\mathbf{d}} + \mathbf{K}_\text{t}(\mathbf{d}^k)\Delta\mathbf{d}^k &= \mathbf{f}_\text{ext} - \mathbf{f}_\text{int}(\mathbf{d}^k) \, , \\ \mathbf{d}^{k+1} &= \mathbf{d}^k + \Delta\mathbf{d}^k \, . \end{aligned} \tag{9}$$

In this system $\mathbf{M}$ is the mass matrix, $\mathbf{f}_\text{ext}$ is the vector of external forces, $\mathbf{f}_\text{int}$ is the vector of internal forces accounting for internal stresses and $\mathbf{K}_t$ is the tangent stiffness matrix being defined as the derivative of $\mathbf{f}_\text{int}$ with respect to $\mathbf{d}$. Time discretization typically uses the Generalized-α method but a quasi-static integration is also available when inertia terms are negligible. The thermo-mechanical coupling is performed by an explicit staggered integration scheme where the thermal part is solved after the mechanical part on each time step. Re-evaluation of the internal stresses after the thermal step is possible for problems that are highly driven by thermal effects.

The structural part of the problem is also solved using the open-source code GetDP [38, 39] that is a free linear Finite Element software and a general environment for the treatment of discrete problems. In GetDP, Equation (7) can

only be treated as a linear problem involving small deformations/displacements and linear elastic materials. Thus the discretized equations for the structural motion take the standard linear form:

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{K}\mathbf{d} = \mathbf{f}_{\text{ext}} \,, \tag{10}$$

where $\mathbf{K}$ is the constant stiffness matrix of the system. Time integration is performed using the Newmark method.

Finally, simpler coupled fluid-structure problems involve the motion of non-deformable solids for which the dynamics is constrained using linear stiffness and damping. Such a model is implemented in an in-house rigid body integrator code also based on the Generalized-α method for time integration.

2.3. Coupling conditions

The following coupling formulation expresses how the fluid (subscript "f") and solid (subscript "s") domains are coupled at their common interface Γ through continuity boundary conditions on the displacement $\mathbf{d}^\Gamma$ and the load $\mathbf{t}^\Gamma$,

$$\begin{aligned} \mathbf{d}_{\text{f}}^\Gamma &= \mathbf{d}_{\text{s}}^\Gamma \,, \\ \mathbf{t}_{\text{f}}^\Gamma &= -\mathbf{t}_{\text{s}}^\Gamma \,, \end{aligned} \tag{11}$$

where the load on the fluid side is given by $\mathbf{t}_{\text{f}} = -p\mathbf{n}_{\text{f}} + \boldsymbol{\tau}\mathbf{n}_{\text{f}}$, with p the pressure and $\boldsymbol{\tau}$ the viscous stress, and the load on the solid side by $\mathbf{t}_{\text{s}} = \boldsymbol{\sigma}\mathbf{n}_{\text{s}}$. The normal unit vectors $\mathbf{n}_{\text{f}}$ and $\mathbf{n}_{\text{s}}$ are both pointing outwards from their respective domains. When thermal coupling is taken into account, additional continuity relations on the temperatures and heat fluxes are considered:

$$\begin{aligned} T_{\text{f}}^\Gamma &= T_{\text{s}}^\Gamma \,, \\ (\lambda_{\text{f}} \nabla T_{\text{f}})^\Gamma &= (\lambda_{\text{s}} \nabla T_{\text{s}})^\Gamma \,, \end{aligned} \tag{12}$$

where λ is the thermal conductivity.

The coupled problem can be solved using a Dirichlet-Neumann approach [40], as illustrated below for mechanical coupling; the technique can be easily extended to thermal coupling. Introducing a Dirichlet nonlinear operator $\mathcal{F}$ that

computes the fluid loads from a given fluid interface displacement,

$$\mathbf{t}_{\mathrm{f}}^{\Gamma} = \mathcal{F}(\mathbf{d}_{\mathrm{f}}^{\Gamma}) \,, \tag{13}$$

and a Neumann nonlinear operator $\mathcal{S}$ that computes the solid interface displacement as a function of the solid loads,

$$\mathbf{d}_{\mathrm{s}}^{\Gamma} = \mathcal{S}(\mathbf{t}_{\mathrm{s}}^{\Gamma}) \,, \tag{14}$$

Equation (11) can be formulated as a fixed-point problem [41]:

$$\mathbf{d}^{\Gamma} = \mathcal{S}\left(-\mathcal{F}(\mathbf{d}^{\Gamma})\right) \,, \tag{15}$$

where $\mathbf{d}^{\Gamma}$ is the displacement common to both the solid and fluid interfaces. Section 3 presents the flexible implementation of CUPyDO that solves this fixed-point problem by considering $\mathcal{F}$ and $\mathcal{S}$ as black-box tools representing generic fluid and solid solvers. The state variables of each solver depend on its intrinsic boundary conditions in addition to the conditions imposed at the coupling interface.

Finally, the mesh within the fluid domain must be adapted to accommodate the deformation/displacement of the solid. The mesh morphing used with the ALE formulation can be expressed as

$$(\mathbf{x}_{\mathrm{f}}^{\Omega}, \mathbf{v}_{\mathrm{f}}^{\Omega}) = \mathcal{M}(\mathbf{d}_{\mathrm{f}}^{\Gamma}) \,, \tag{16}$$

where $\mathbf{x}_{\mathrm{f}}^{\Omega}$ and $\mathbf{v}_{\mathrm{f}}^{\Omega}$ are the position and velocity of the mesh points, respectively, and $\mathcal{M}$ a mesh deformation operator. The combination of the $\mathcal{F}$, $\mathcal{S}$ and $\mathcal{M}$ operators may be referred to as a three-field problem [42]. In this paper, the mesh morphing step is considered as a fluid solver task so that this procedure is implicitly included in Equation (15).

3. IMPLEMENTATION OF THE COUPLING ENVIRONMENT

The implementation of the computational tool CUPyDO designed to solve the fixed-point equation (15) using a partitioned approach is described hereafter. CUPyDO uses the Python language in order to interface independent solvers,

usually written in a compiled language, in one single and integrated framework such that the solvers can be intuitively synchronized and data can be exchanged between each other. The coupler provides a ready-to-use coupling algorithm as well as interpolation capabilities for non-matching fluid-structure interface meshes. Parallel functionalities are also available, based on communication (collective or point-to-point) between the processes of each solver involved in the computation. Python bindings for the Message Passing Interface (MPI) protocol as well as Python bindings for the PETSc library, used for all parallel linear algebra operations mainly required for the mesh interpolation step, are available. Point searches and filtering are performed using binary trees for efficient computation of nearest neighbors during the mesh mapping.

3.1. Coupling methodology and Python wrapping

In order to ensure the highest level of flexibility, the coupling of the solvers is based on an abstracted black-box approach that is achieved through a modular and high-level implementation of the coupling environment using the Python programming language. Modules and functionalities of the coupled solvers are wrapped in a Python layer that behaves as a driving and communicating channel for CUPyDO. The wrapping procedure is easily performed using the Simplified Wrapper and Interface Generator (SWIG) tool [43]. SWIG is able to interface any function or object defined in the core code of the solvers with Python by performing an additional but not intrusive compilation step. The generated Python wrapper plays the role of a scripting API without using brute code translation nor interfering with the libraries and executables created during the basic compilation. A schematic example is proposed below based on C++, which is the language used by the fluid and solid solvers involved in this paper. However, the procedure can be extended to any compiled language, including the widely used Fortran language. Considering the following C++ code defining a simple object:

```
1 //File: myobject.h
2 //C++ code for the definition of MyObject class
3 #pragma once
4
```

```
#include <string>
//If required, headers from the core code can be used
#include "corecode.h"

class MyObject{
  double alpha;
  std::string tag;
public:
  MyObject(std::string const& val_tag);
  ~MyObject();
  void set_alpha(double val_alpha);
  double get_alpha() const;
  void run();
};
```

and wrapping this code using SWIG will create a Python module, e.g. MyModule, that could be used with the following code:

```
#Python code that uses the wrapper of MyObject
import MyModule

pyobject = MyModule.MyObject("Put a tag here")
pyobject.set_alpha(5.0)
data = pyobject.get_alpha()
pyobject.run()
```

This methodology is the basic principle used by the coupler CUPyDO to synchronize the sub-systems and perform communications. For data exchange and to call the required functionalities, CUPyDO directly interacts with the solvers through their wrappers as if they were simple Python objects. These communications do not involve any file I/O. The wrapped functionalities of each solver can thus be easily and intuitively managed in Python while the critical and computationally intensive calculations are performed by the native solvers under their own language. Particular attention may be paid to the fact that the solvers are not driven through basic OS system calls. This allows the coupled simulation to avoid redundant and time-consuming pre-processing operations, such as mesh construction and configuration reading, at each solver call.

Another example of the wrapping process and the interaction between the solvers and the coupling environment is given in Figure 1 with a broader point of view on the procedure that shows all the steps from the core code to a simplified generic Python coupling environment by including the SWIG compilation and the Python wrapper generation.

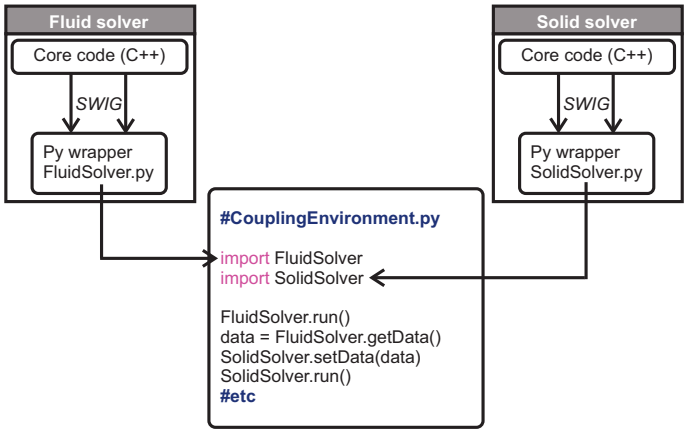

Figure 1: Schematic illustration of a coupling environment and its interaction with the respective fluid and solid Python wrappers.

This way of using a Python framework as a coupling environment allows the users and the developers to reach the largest level of flexibility for performing top level tasks such as managing the coupled solvers and communicating the data. First, those data can be expressed under friendly Python-oriented formats such as lists or dictionaries, or with *numpy* [44] arrays for larger data sets on which computational operations have to be performed. SWIG allows those Python types to be interfaced with classical and efficient static C/C++ arrays (pointers) or C++ `std::vector`, again taking C/C++ as an example. Secondly, using a Python wrapping methodology is less intrusive than compiling the coupled code with an external library or an API adapter coming from the coupler. Thirdly, the Python wrapper can be generated as a generic interfacing layer without being restricted to FSI coupling purposes. Finally, the coupling with commercial codes is technically conceivable since several of them, such as Abaqus [45], are already designed with a Python interface.

3.2. Overall coupling architecture

The coupling Python environment CUPyDO is an object-oriented code whose architecture is summarized in Figure 2 where the main classes are represented with white boxes. The framework is divided into three distinct layers: *Utilitiy* (U), *Core* (C) and *Interface* (I). The *Utility* layer defines common functionali-

ties such as MPI communication functions based on the *mpi4py* [46] wrapping module. The `Interface data` class is designed to handle in parallel the data that have to be exchanged at the fluid-structure interface between the solvers. Equivalently, the `Interface matrix` class is used to define and construct in parallel the interpolation matrix used for the mapping of non-matching meshes (Section 3.5). Those classes are derived from the *petsc4py* [47] wrapping module for the PETSc library so that they can be used in every parallel algebraic operation such as `matrix(data)-data` (scalar) products or `matrix-data` linear systems. In this case, the `Linear solver` class, which is an interface to the Krylov-type iterative solvers from PETSc, is used.

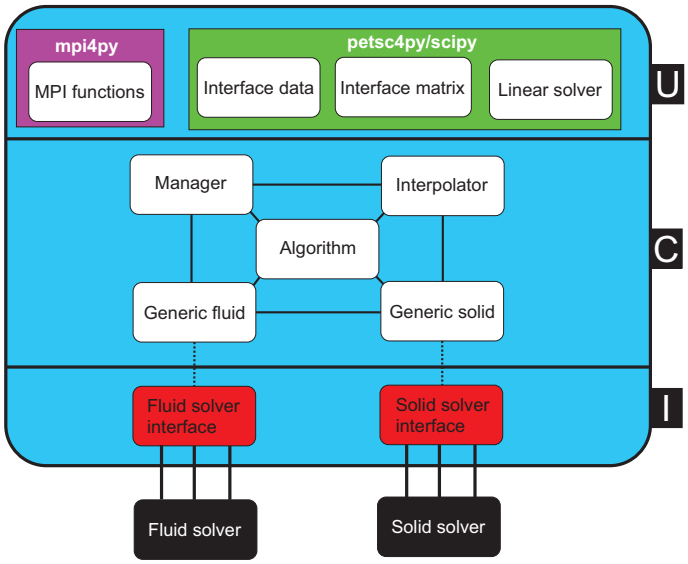

Figure 2: Overview of the implemented architecture of CUPyDO.

The *Core* layer is the central part of the coupling environment where the main classes are defined according to different tasks such as the management of the MPI partitioning and the communication network, the interpolation of the fluid-structure interface meshes, the coupling algorithm and the interfacing with the solvers. The `Manager` class is the first class to be instantiated and is designed to build the network describing the MPI partitioning of each solver. For example, this will identify the processes on which the fluid and/or

solid solvers are running and, among all those processes, it will distinguish the subset of processes that effectively own fluid-structure interface nodes. Storing the number of interface nodes on each process and identifying the halo nodes (i.e. nodes that support the communication between the MPI partitions of one solver) are also important tasks dedicated to the `Manager`. The `Interpolator` uses the information built by the `Manager` to construct the non-matching mesh interpolant that will be called each time the solvers have to exchange their data from one grid to the other. The coupled solvers are represented in the *Core* layer by `Generic fluid/solid` classes (one fluid and one solid). These are almost pure virtual classes whose purpose is to ensure the flexibility and the compatibility between the solvers and the single coupling environment. Finally, the `Algorithm` class is the central part of the *Core* layer where the coupling algorithm (see next section) is actually implemented. The other classes are synchronized inside the `Algorithm` in order to perform all the coupling tasks such as communication, interpolation, and sub-system computation.

The *Interface* layer is an important part of the coupling environment since it ensures the flexibility of the coupling and the compatibility between the solvers and the coupling environment. It is composed of child interfacing classes (red boxes in Figure 2) that are directly derived from the generic classes of the *Core* layer. First, this inheritance ensures the compatibility between any coupled solver and the central `Algorithm` class. Then the interfacing classes are overloaded with the particular wrapper coming from each coupled solver so that one interfacing class (red box) is required per coupled solver. The interfacing class plays the role of a plugin layer and thus ensures the flexibility of the coupling. Consequently, the individual Python wrappers of the solvers do not have to be designed while seeking a deep compatibility with the coupling environment. Also, modifying the coupling environment, e.g. for maintenance or improvements, does not need to affect the individual Python wrappers and, reversely, any deep change in the core code of the coupled solvers has no impact on the central coupler. Templates for interfacing classes are provided by CUPyDO to users wishing to couple their solvers providing a Python wrapper can

be generated. Users then have the possibility to overload the provided templates with the wrapped features of their solvers in order to ensure compatibility with the coupler.

This architecture allows the user to launch the coupled simulations with a very simple Python script and a minimal amount of code, as illustrated by the code below:

```
#Schematic launching script for a coupled simulation with CUPyDO
import cupydo

args['fluid'] = list()
args['solid'] = list()
args['man'] = list()
args['int'] = list()
args['alg'] = list()

# Initialization of fluid solver
import fluidSolverInterface
fluidSolver = fuidSolverInterface.FluidSolverConstructor(args['
    fluid'])

# Initialization of solid solver
import solidSolverInterface
solidSolver = solidSolverInterface.SolidSolverConstructor(args['
    solid'])

# Initialization of manager
manager = cupydo.Manager(fluidSolver,solidSolver,args['man'])

# Initialization of interpolator
interpolator = cupydo.Interpolator(manager,fluidSolver,
    solidSolver, args['int'])

# Initialization of algortihm
algorithm = cupydo.Algorithm(manager,fluidSolver,solidSolver,
    interpolator,args['alg'])

# Run the coupled simulation
algorithm.run()
```

The fluid and solid solvers are first instantiated with their own parameters, e.g. configuration and mesh files, followed by the manager, interpolator and algorithm, in this order to respect the dependencies. The coupled computation is then launched by calling the `run` method of the algorithm class.

3.3. Coupling algorithm

The coupling environment of CUPyDO for solving the FSI problem, represented by Equation (15), is implemented using a partitioned framework. It is

based on a block-Gauss-Seidel (BGS) algorithm [41, 48], that synchronizes the solvers in a strong coupling scheme. After one fluid computation, the loads are transferred to the structural part which is also solved in order to calculate the solid displacements that are defined as new fluid boundary conditions for the next coupling iteration. The coupling conditions on the fluid-structure interface at each time step (for time-marched computations) are met by iterating between the fluid (operator $\mathcal{F}$) and solid (operator $\mathcal{S}$) computations and by exchanging their boundary conditions. The same procedure can be applied for thermal coupling by exchanging temperatures and heat fluxes. These iterations are repeated until a convergence criterion, based on the Euclidean norm of the difference in the structural displacements (or temperatures and heat fluxes) between two successive BGS iterations (index j),

$$||\mathbf{r}_j^\Gamma|| = ||\tilde{\mathbf{d}}_j^\Gamma - \mathbf{d}_{j-1}^\Gamma|| < \varepsilon\,, \tag{17}$$

is met. In Equation (17), ε is a case-dependent dimensional tolerance that is set by the user, $\tilde{\mathbf{d}}_j^\Gamma$ is the computed displacement by the solid solver at the current BGS iteration and $\mathbf{d}_{j-1}^\Gamma$ is the relaxed displacement at the previous BGS iteration.

For cases where the fluid density is close to the structural density, strong interactions between the fluid and solid are expected (added-mass effects). This leads to a slower convergence of the BGS coupling that can even diverge in the most severe cases and an under-relaxation on the computed structural displacement can be applied to stabilize the coupling:

$$\mathbf{d}_j^\Gamma = \mathbf{d}_{j-1}^\Gamma + \omega_j \mathbf{r}_j^\Gamma\,, \tag{18}$$

with $\omega < 1$. Static (constant ω) or dynamic relaxation are both available. In the dynamic case, Aitken's formulation [41] is used to update the relaxation parameter at each coupling iteration:

$$\omega_j = -\omega_{j-1} \frac{(\mathbf{r}_{j-1}^\Gamma)^{\mathrm{T}} \cdot (\mathbf{r}_j^\Gamma - \mathbf{r}_{j-1}^\Gamma)}{||\mathbf{r}_j^\Gamma - \mathbf{r}_{j-1}^\Gamma||^2}\,. \tag{19}$$

For time-marched solutions, Aitken's formula cannot be directly applied at the first BGS iteration. The last calculated ω value at the previous time step is

thus used as initial value for the first BGS iteration. This value can be limited by an upper or lower user-defined boundary. The choice of an upper bound is more conservative from a stability point of view but can lead to a higher number of coupling iterations and thus slower convergence of the iterative procedure. The convergence of the BGS coupling algorithm at each time step can also be improved by using a predictor [49] on the solid displacements at the beginning of a time step i, before the first fluid computation:

$$\mathbf{d}_i^\Gamma = \mathbf{d}_{i-1}^\Gamma + \alpha_0 \, \Delta t \, \dot{\mathbf{d}}_{i-1}^\Gamma + \alpha_1 \, \Delta t \, (\dot{\mathbf{d}}_{i-1}^\Gamma - \dot{\mathbf{d}}_{i-2}^\Gamma) \,, \tag{20}$$

in which $\alpha_0 = 1$ and $\alpha_1 = 0.5$ for a second-order predictor. The predicted value is then communicated to the fluid solver as an initial guess of the fluid-structure interface position. The overall time-dependent coupling algorithm is illustrated in Figure 3. Note that an explicit (weak) coupling can be achieved if only one coupling iteration is performed at each time step.

In order to track the motion of the fluid-structure interface, the deformation of the fluid volume mesh is usually chosen over a complete and costly remeshing. Deforming the grid also allows us to conserve its topology. This mesh morphing task is a major step in the coupling algorithm, especially for large structural deformations where low quality fluid cells may appear leading to a low quality fluid solution or even the divergence of the computation. In this architecture, the fluid volume mesh deformation step is considered as an intrinsic feature of the coupled fluid solver in line with its own ALE implementation (as described in Section 2.1). Since it is expected that the fluid part of the problem can deform its own volume mesh with its own method, no mesh deformation technique is currently implemented in CUPyDO.

3.4. Conjugate heat transfer capabilities

As already mentioned, thermal transfer between the coupled solvers is also possible. In addition to loads and displacements, temperatures and heat fluxes can be communicated through the coupler. Four thermal coupling schemes are available depending on the direction of the data transfer [50, 51]: Temperature

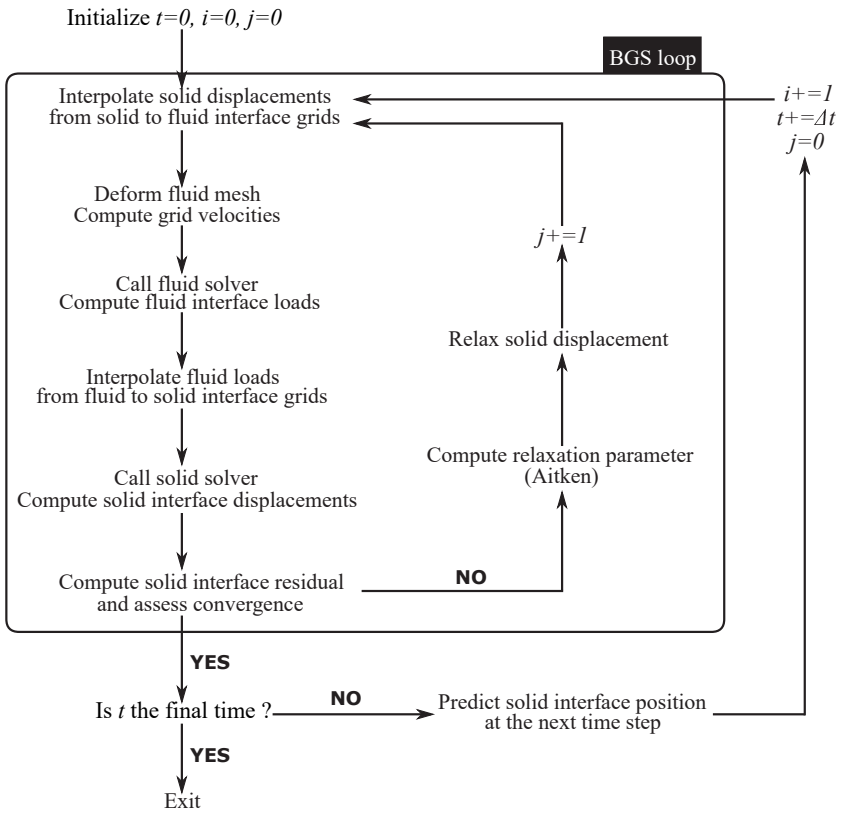

Figure 3: Time-marching coupling algorithm based on a block-Gauss-Seidel scheme (i is the time iterator and j is the FSI iterator).

Forward Flux Back (TFFB), Flux Forward Temperature Back (FFTB), Heat transfer coefficient Forward Temperature Back (hFTB) and Heat transfer coefficient Forward Flux Back (hFFB). Those are defined from a fluid-side point of view and are depicted in Figure 4.

The first two schemes directly exchange temperatures T and heat fluxes $\mathbf{q}$ at the fluid-structure interface. In this case, stability criteria are found to be dependent on the Biot number of the coupled problem [51]. The TFFB scheme is stable for Bi > 1 whereas the FFTB scheme is stable for Bi < 1. The last two schemes use a Robin boundary condition on the solid side. They are based on a user-defined parameter h_c being a numeric, i.e. with no true physical meaning, convective heat transfer coefficient. This coefficient is used as

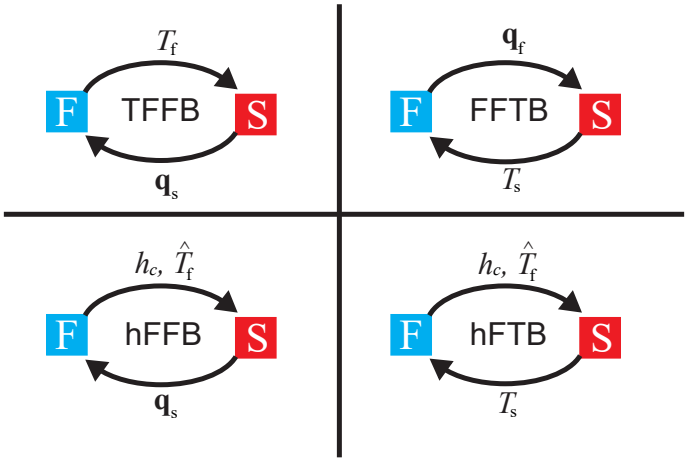

Figure 4: Illustration of the available thermal coupling schemes for CHT applications.

a relaxation parameter that affects the convergence rate of the coupled solution. On the fluid interface, the convective heat flux can be written as follows, by using the coefficient h_c and defining an equivalent fluid temperature $\hat{T}_f$:

$$q_f^\Gamma = h_c \left(T_f^\Gamma - \hat{T}_f \right) , \tag{21}$$

where

$$q_f^\Gamma = \mathbf{q}_f^\Gamma \cdot \mathbf{n}_f^\Gamma . \tag{22}$$

From these expressions, the equivalent fluid temperature is calculated and then communicated to the solid domain. Equation (21) is written on the solid domain to get the heat flux that is imposed as a solid boundary condition:

$$q_s^\Gamma = h_c \left(T_s^\Gamma - \hat{T}_f \right) . \tag{23}$$

The thermal coupling uses the same algorithm as for the mechanical problem without Aitken relaxation in such a way that only the additional thermal data have to be communicated through the interface. The user is allowed to choose between pure mechanical, pure thermal or mixed mechanical-thermal couplings. Independent coupling tolerances are defined by the user for each physics.

3.5. Non-matching fluid-structure interface meshes

In a partitioned coupling approach solid and fluid meshes are likely to be created independently of each other for optimality in resolving different physics that impose different stretching constraints. Consequently, there is no guarantee that the boundary discretization at the fluid-structure interface consists in matching meshes. Non-matching geometries may also be used. For example, one may represent a wing with its true geometry in the fluid domain so that the fluid loads are correctly recovered, but this same wing could be represented as an equivalent structural box in the solid domain. In both cases, the data transferred between the two solvers must be interpolated from one grid to the other. This is equivalent to defining a new operator $\mathcal{I}_{\mathrm{s}}^{\mathrm{f}}$ that maps the displacements of the solid interface mesh onto the fluid interface mesh during the communication step of the BGS algorithm,

$$\mathbf{d}_{\mathrm{f}}^{\Gamma} = \mathcal{I}_{\mathrm{s}}^{\mathrm{f}}(\mathbf{d}_{\mathrm{s}}^{\Gamma})\,, \tag{24}$$

or, analogously, an operator $\mathcal{I}_{\mathrm{f}}^{\mathrm{s}}$ that maps the load from the fluid to the solid,

$$\mathbf{t}_{\mathrm{s}}^{\Gamma} = \mathcal{I}_{\mathrm{f}}^{\mathrm{s}}(\mathbf{t}_{\mathrm{f}}^{\Gamma})\,. \tag{25}$$

The same operators can be defined for interpolation of temperatures and heat fluxes. These operators can be expressed as simple linear algebraic interpolation matrices [52]. Equation (24) for the displacement of the fluid-structure interface can be rewritten as

$$\mathbf{d}_{\mathrm{f}}^{\Gamma} = \mathbf{H}\mathbf{d}_{\mathrm{s}}^{\Gamma}\,, \tag{26}$$

and a conservative interpolation is used for the loads so that Equation (25) becomes

$$\mathbf{t}_{\mathrm{s}}^{\Gamma} = \mathbf{H}^{\mathrm{T}}\mathbf{t}_{\mathrm{f}}^{\Gamma}\,. \tag{27}$$

Conservative interpolation assumes that energy is conserved through the fluid-structure interface [53] by writing the equilibrium of virtual work acting on the fluid and solid sides:

$$\delta W_s = \left(\mathbf{t}_{\mathrm{s}}^{\Gamma}\right)^{\mathrm{T}} \delta\mathbf{d}_{\mathrm{s}}^{\Gamma} = \left(\mathbf{t}_{\mathrm{f}}^{\Gamma}\right)^{\mathrm{T}} \delta\mathbf{d}_{\mathrm{f}}^{\Gamma} = \delta W_f\,, \tag{28}$$

such that only one matrix, i.e. a one-sided mapping (solid to fluid), is needed between the fluid and solid interface grids. For a consistent interpolation [53], matrices for the displacements and for the loads are built separately and need a two-sided mapping (solid to fluid and fluid to solid). Consistent mapping is used for thermal data interpolation.

The interpolation matrix $\mathbf{H}$ is defined according to the technique used to map the interface nodes from one grid to the other. Nearest neighbor interpolation is the cheapest technique and results in a boolean structure for the interpolation matrix but also leads to a very poor interface reconstruction when the difference on the discretization of the two domains becomes significant. A stair-shaped fluid interface is recovered since several fluid nodes may have the same displacement coming from one particular solid node. However this method is perfectly suited for matching mesh mapping. Projection methods consist in projecting the fluid nodes onto the structural mesh and using the shape functions of the structural elements to define the elements of $\mathbf{H}$. Topological information is required to solve the projection problem leading to a lack of flexibility if this is not directly available from the coupled solvers. Moreover, the projection step is prone to robustness issues when the discretization mismatching is again significant.

The interpolation technique implemented in the present coupling architecture is a point-match method based on Radial Basis Function (RBF) interpolation [54, 52, 55]. This approach is said to be "meshless" because no topological information is required, thus conserving coupling flexibility. The parallelization of the method is also easier and the accuracy/cost ratio can be nicely tuned by the user with minimal effort.

In the context of RBF, the interpolation of a quantity of interest reads

$$s(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i \phi \left(||\mathbf{x} - \mathbf{x}_i|| \right) + p(\mathbf{x}) \,, \tag{29}$$

where the position of an interface mesh point is given by the vector $\mathbf{x} = [x, y, z]^{\mathrm{T}}$, N is the number of points, $p(\mathbf{x})$ is a polynomial and ϕ are basis functions of the Euclidean distance. The coefficients α_i and the coefficients of the polynomial

are determined by requiring an exact recovery of the function, for example the structural nodal displacements,

$$\mathbf{d}_{\mathrm{s}}^{\Gamma}(\mathbf{x}_j) = \mathbf{d}_{\mathrm{s},j}^{\Gamma} \tag{30}$$

and the additional requirements

$$\sum_{i=1}^{N} \alpha_i q(\mathbf{x}) = 0 \tag{31}$$

for any polynomial q with a degree less than or equal to that of polynomial p. The degree of the polynomial p depends on the choice of the basis function ϕ. A unique interpolant is given if the basis function is a conditionally positive definite function; if it is of order $m \leq 2$, a linear polynomial p,

$$p(\mathbf{x}) = \beta_0 + \beta_x x + \beta_y y + \beta_z z \,, \tag{32}$$

can be chosen [54]. Using a linear polynomial guarantees that any rigid body motion of the fluid-structure interface will be recovered. Discretely, Equations (29) and (31) are written for the structural displacement in a matrix form as (for the x-dimension, similar expressions in the other dimensions)

$$\begin{bmatrix} \mathbf{d}_{s,x}^{\Gamma} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{C}_{\mathrm{ss}} & \mathbf{P}_{\mathrm{s}} \\ \mathbf{P}_{\mathrm{s}}^{\mathrm{T}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_x \\ \boldsymbol{\beta}_x \end{bmatrix} \,, \tag{33}$$

where the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ vectors contain the coefficients α_i and the coefficients of the polynomial p, respectively. The matrix $\mathbf{C}_{ss}$ is a $N_s \times N_s$ matrix containing all the evaluations of the basis functions such as $\mathbf{C}_{ss}(i,j) = \phi\left(||\mathbf{x}_{\mathrm{s}_i} - \mathbf{x}_{\mathrm{s}_j}||\right)$ and the matrix $\mathbf{P_s}$ is a $N_s \times 4$ matrix whose rows are defined by $\begin{bmatrix} 1 & x_{\mathrm{s}_j} & y_{\mathrm{s}_j} & z_{\mathrm{s}_j} \end{bmatrix}$. A similar expression can be used for the fluid displacement, for which Equation (29) writes

$$\mathbf{d}_{\mathrm{f},x}^{\Gamma} = \begin{bmatrix} \mathbf{C}_{\mathrm{fs}} & \mathbf{P}_{\mathrm{f}} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}_x \\ \boldsymbol{\beta}_x \end{bmatrix} \tag{34}$$

In this case $\mathbf{C}_{\mathrm{fs}}$ and $\mathbf{P}_{\mathrm{f}}$ contain the evaluation of the basis functions based on distances computed between fluid and solid nodes and the coordinates of the fluids nodes, respectively.

Combining Equations (33) and (34) enables us to express the interpolation matrix $\mathbf{H}$ as being the first $N_\mathrm{f} \times N_\mathrm{s}$ block of

$$\widetilde{\mathbf{H}} = \begin{bmatrix} \mathbf{C}_\mathrm{fs} & \mathbf{P}_\mathrm{f} \end{bmatrix} \begin{bmatrix} \mathbf{C}_\mathrm{ss} & \mathbf{P}_\mathrm{s} \\ \mathbf{P}_\mathrm{s}^\mathrm{T} & \mathbf{0} \end{bmatrix}^{-1} . \tag{35}$$

Similarly, we write the transposed interpolation matrix that gives the conservative interpolation of the fluid loads on the solid mesh as being the first $N_\mathrm{s} \times N_\mathrm{f}$ block of

$$\widetilde{\mathbf{H}}^\mathrm{T} = \begin{bmatrix} \mathbf{C}_\mathrm{ss} & \mathbf{P}_\mathrm{s} \\ \mathbf{P}_\mathrm{s}^\mathrm{T} & \mathbf{0} \end{bmatrix}^{-\mathrm{T}} \begin{bmatrix} \mathbf{C}_\mathrm{fs} & \mathbf{P}_\mathrm{f} \end{bmatrix}^\mathrm{T} . \tag{36}$$

The construction of the different blocks requires nothing else than node coordinates and distance computations. The sub-matrices are assembled at the beginning of the overall simulation but the matrix $\mathbf{H}$ and its transpose are not computed explicitly since they require matrix inversion. This operation might be costly and inaccurate. Moreover, a direct inversion of the matrix does not take advantage of the potential sparsity of the matrix to be inverted if basis functions defined on a radial compact (see below) are used for the interpolation. Consequently the system (33) is solved using the FGMRES iterative solver of the PETSc library and the product (34) is computed at each communication of the coupled simulation. A FGMRES approach was chosen as default because it is known to provide good convergence and has proved to be robust in most cases, but any other PETSc solver could otherwise be used.

The intrinsic behavior of the interpolation algorithm is determined by the choice of the basis functions. In CUPyDO, two commonly used basis functions are available, namely the $\mathcal{C}^2$ function with radial compact support of size r,

$$\phi(||\cdot||) = \left(1 - \frac{||\cdot||}{r}\right)_+^4 \left(4\frac{||\cdot||}{r} + 1\right) , \tag{37}$$

where the subscript $+$ indicates that only the positive part of the expression is taken into account, and the Thin Plate Spline (TPS) with global support,

$$\phi(||\cdot||) = ||\cdot||^2 \log(||\cdot||) . \tag{38}$$

The $\mathcal{C}^2$ function gives a sparse interpolation system for which the sparsity level is dictated by the value of the radius r. A large value of the radius yields a more accurate interpolation but also a larger system to store in memory and to solve. Conversely, small values of the radius imply sparse and light systems with lower accuracy. The value of the radius is constant and dimensional. It is fixed by the user as a case-dependent configuration parameter. There is no specific rule that defines an optimal value for the radius but choosing an excessively small value could lead to inaccurate interface reconstruction. The global TPS function was shown to give accurate interpolations but automatically involves full systems to be stored and solved [52].

3.6. Parallelization

CUPyDO is developed for large systems requiring a parallelized implementation with MPI. In the framework of a partitioned architecture, the parallelization is not straightforward and keeping a high level of flexibility while conserving good parallel scalability is challenging. Two types of communications are considered: intra- and inter-communications. Intra-communications refer to the communications between the processes belonging to one of the coupled solvers. These communications usually depend on the intrinsic parallelization of the solvers and can be treated as black-box functionalities. This abstracted approach also allows CUPyDO to use pure serial solvers. Inter-communications refer to the communications between processes across the coupling algorithm. These communications are typically used to exchange data in parallel between the coupled solvers on top of the primary communication mechanism based on the wrapping methodology.

The most flexible inter-communication approach would be to use a sequential coupling interface. Collective communications are used to gather the data from the parallel solvers on one single core and the coupling computations, such as fluid-structure mesh interpolation, are run in serial. This obviously leads to a severe bottleneck due to the loss of the fluid-structure interface partitioning and the serialization of some computation-intensive tasks. In the coupling ar-

chitecture of CUPyDO, parallel coupling is achieved using a re-partitioning of the fluid-structure interface from a local distribution to a global distribution. This re-partitioning step is used to perform coupling calculations in parallel with adequate load balancing on the fluid-structure interface nodes. The parallel coupling computations are performed using the PETSc library by interfacing its Python bindings directly into the coupling architecture (as mentioned in Section 3.2). Typically, this allows us to construct the interpolation matrices and solve the associated system of equations in parallel.

The parallel coupling scheme is depicted in Figure 5 for data interpolation. The re-partitioning of the fluid-structure interface is directly performed by the PETSc library through the `Interface data` class that is globally assembled in parallel from the processes owning the nodes belonging the the fluid-structure interface (filled boxes in Figure 5). This is called local-to-global mapping. Fluid-structure interface nodal data are then interpolated, with parallel system solutions and parallel matrix-vector products, in the global space and then redistributed to their respective solver instances using reverse mapping (from global to local). Reverse mapping is performed by first gathering all the data on the master thread and then distributing the data with peer-to-peer communications. A direct peer-to-peer communication strategy that avoids gathering of the data is part of ongoing work. In order to build the parallel interpolation matrix, represented in green in Figure 5, each partition of the interface solid (fluid) mesh is sent to the partitions of the fluid (solid) interface mesh through several communication rounds. For each round, the receiving fluid (solid) partitions compute the elements of $\mathbf{C}$ and $\mathbf{P}$ using the RBF mapping. In case of local RBF, the mapping is enhanced by building K-D trees of the received mesh partitions in order to filter the points that are outside the specified radius, thus reducing the amount of data that have to be treated.

The parallel implementation of the coupler has been developed to take into account the heterogeneous distribution of each coupled solver. Most of the time, the fluid domain will require many more processors than the solid one. Consequently, different numbers of processors should be allocated for the fluid/solid

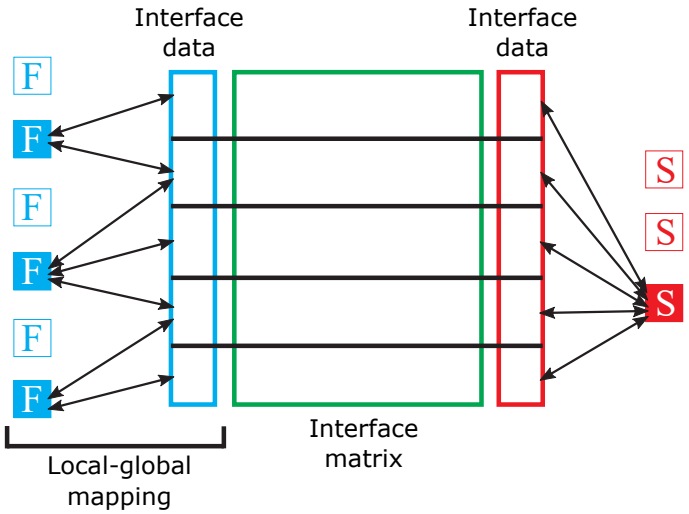

Figure 5: Parallel distribution mapping and parallel fluid-structure interface data interpolation (blue for fluid side and red for solid side). Each box represents a process on which the fluid or solid solver is instantiated. Filled boxes represents processes owning fluid-structure interface nodes.

solvers. This segregation can be achieved by defining specific groups of processes, that are subsets of the total amount of processes, on which the solver will be instantiated. Consequently, each process can instantiate a fluid, a solid or both fluid/solid instances. When there is no intersection between the fluid and solid subsets of processes (the processes instantiate either a fluid or a solid, never both), simultaneous computations where the fluid and solid solvers are running at the same time could be considered. Although the parallel framework of CUPyDO is compatible with such a distribution of processes, there is no coupling algorithm currently implemented to tackle this case.

Additionally, other tasks that are purely related to the coupling, such as residual computation, interface relaxation or interface prediction, are also distributed in the global partitioning space. This ensures a fully parallelized coupling algorithm.

4. RESULTS

The implementation of CUPyDO described in the previous section has been used to solve several FSI test cases in order to demonstrate its accuracy, flexibility and robustness. Depending on the test case, different solid solvers are used whereas the fluid solver is always SU2. Additionally, different functionalities of the coupler are highlighted, such as the interpolation of non-matching meshes or under-relaxation of the BGS coupling.

4.1. Isogai wing section

The coupling between SU2 and the rigid body integrator is tested using the classical Isogai wing section aeroelastic case (case A) [56, 57]. This test case represents the dynamics of the outboard portion of a swept-back wing in the transonic regime. The airfoil is a symmetric NACA 65a010 profile with chord $c = 2b$. The two-degree-of-freedom structural model is shown in Figure 6. The displacement h of the elastic axis is positive downwards and the pitch angle α is positive clockwise. The static unbalance S is defined as the product of the airfoil mass m with the distance $x_{\mathrm{CG}} - x_{\mathrm{f}}$ between the center of gravity and the elastic axis. The structural restoring force is provided by a spring-dashpot system with stiffnesses K_h and K_α and damping coefficients C_h and C_α for the plunging and pitching mode, respectively.

The equations of motion for this aeroelastic system can be written as [58]

$$\begin{aligned} m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L\,, \\ S\ddot{h} + I_{\mathrm{f}}\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M\,, \end{aligned} \tag{39}$$

where I_{f} is the moment of inertia of the airfoil around the elastic axis, L the aerodynamic lift (positive upwards) and M the aerodynamic moment with respect to the elastic axis (positive clockwise). The overall system is characterized by several non-dimensional parameters, i.e., the normalized static unbalance $\chi = S/mb$ and moment of inertia $r_\alpha^2 = I_{\mathrm{f}}/mb^2$, the plunging and pitching damping ratios $\eta_h = C_h/2\sqrt{K_h m}$ and $\eta_\alpha = C_\alpha/2\sqrt{K_\alpha I_{\mathrm{f}}}$, the mass ratio $\mu = m/\pi\rho_\infty b^2$ where ρ_∞ is the free-stream fluid density, and the natural

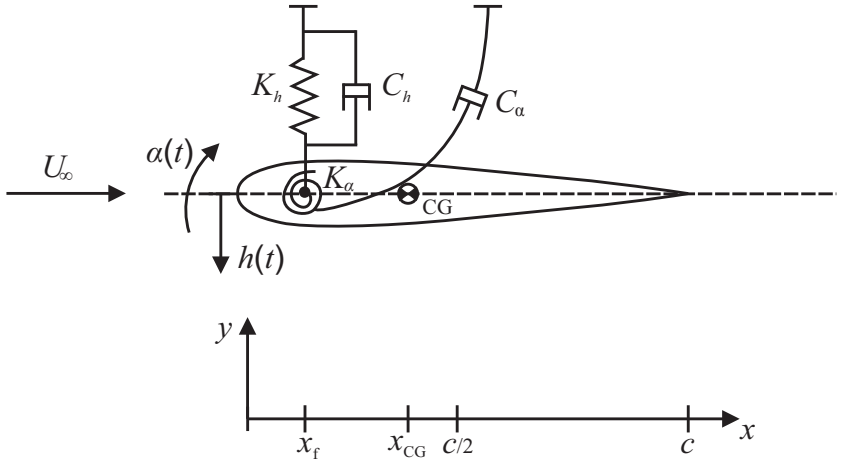

Figure 6: Schematic of a two-degree-of-freedom pitching-plunging airfoil aeroelastic model.

frequency ratio $\overline{\omega} = \omega_h/\omega_\alpha$ where $\omega_h = \sqrt{K_h/m}$ and $\omega_\alpha = \sqrt{K_\alpha/I_\mathrm{f}}$ are the natural frequencies of the uncoupled system. The parameters for the Isogai test case are $\chi = 1.8$, $r_\alpha = 1.865$, $\overline{\omega} = 1$ and $\mu = 60$. There is no structural damping, i.e., $C_h = C_\alpha = 0$. The elastic axis is placed in front of the airfoil at a distance $x_\mathrm{f} = -b$ from the leading edge and the natural pitching frequency is here $\omega_\alpha = 100$ rad/s.

The Euler equations are solved in the transonic regime on the fluid domain which is discretized by a structured O-mesh of 21760 cells (68× 320 for the radial and circumferential direction respectively) with a small stretching from the airfoil surface to the outer boundary. The external domain is circular and extends up to $25c$ from the airfoil in each direction. The simulation is performed starting from uniform flow and an initial airfoil pitch angle $\alpha_0 = 0.0174$ rad (1^o). The time step is set to $\Delta t = 0.0016$ s which corresponds to 39 time steps per period of the uncoupled pitch mode. Because of the high mass ratio of the coupled system, low added-mass effects are expected, thus no relaxation is used in the coupling algorithm. Three BGS iterations per time step are typically required to achieve a coupling tolerance of 10^{-6} m (10^{-4} times the displacement of the center of gravity associated to the initial perturbation) on the structural displacement.

Several FSI simulations at different transonic free-stream Mach numbers ($M_\infty = 0.7 - 0.9$) are performed with variable speed index

$$V^* = \frac{U_\infty}{b\omega_\alpha\sqrt{\mu}}\,, \tag{40}$$

where U_∞ is the free-stream velocity, in order to predict the flutter point. Flutter is identified as the point for which the damping rate of the system's dynamic response is zero. For each speed index, the damping coefficient ζ is computed from the logarithmic decrement of the time response on the pitch and plunge degrees of freedom. The next speed index is determined by interpolating/extrapolating (least square fitting) the damping coefficients previously computed and plotted on a $\zeta - V^*$ diagram.

The computed flutter speed indices V_f^* are compared to values from the literature [59, 60, 61, 62], in Figure 7. The best approximation curve (spline) is a representation of the flutter boundary, i.e. the limit between the stable (under the flutter point) and unstable (beyond the flutter point) regions. It can be seen that the "transonic dip" and the typical "S-shape" of the boundary for M_∞ between 0.7 and 0.9 are both well predicted.

4.2. VIV of a flexible cantilever in the wake of a square cylinder

The study of the flexible cantilever attached to the downstream side of a perfectly rigid square cylinder is a classical two-dimensional benchmark test case for FSI [49]. The geometry of the computational domain is described in Figure 8. In this case $H = 0.01$ m. The physical properties of the solid and fluid are summarized in Table 1. The uniform incoming flow velocity is $U_x = 0.513$ m/s, which corresponds to a Reynolds number $\mathrm{Re} = U_x H/\nu_\mathrm{f} = 333$. The top and bottom sides of the domain are modeled as inviscid walls whereas no-slip conditions are imposed on solid boundaries (cylinder and cantilever).

The velocity and Reynolds number are such that an unsteady laminar Von Karman vortex street is generated in the wake of the cylinder with a shedding frequency close to the first bending frequency of the flexible cantilever. Therefore, the vortical structure of the wake generates harmonic aerodynamic loads that induce periodic oscillations of the flexible cantilever.

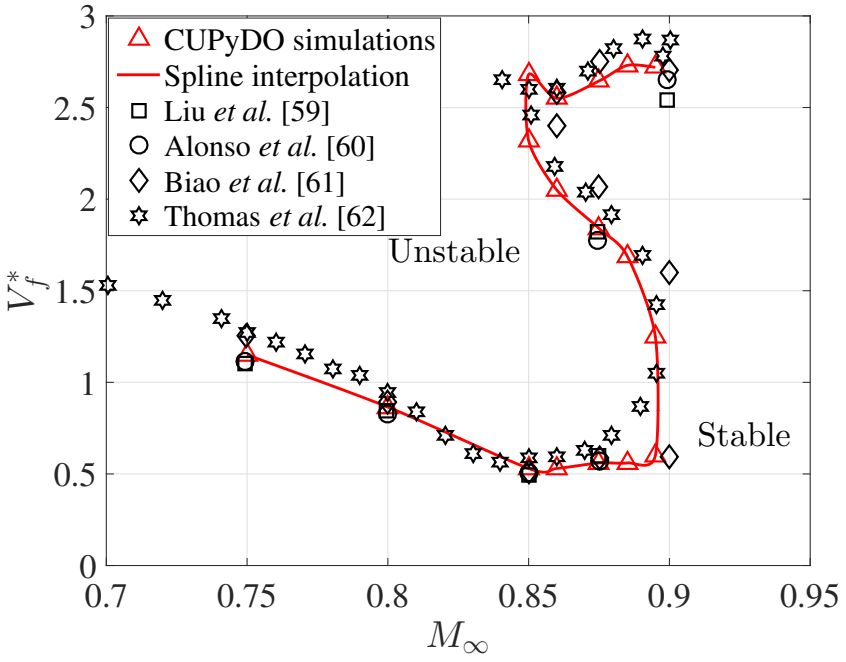

Figure 7: Flutter speed index as a function of the free-stream Mach number for the Isogai wing section. Comparison between current computations and numerical results from the literature.

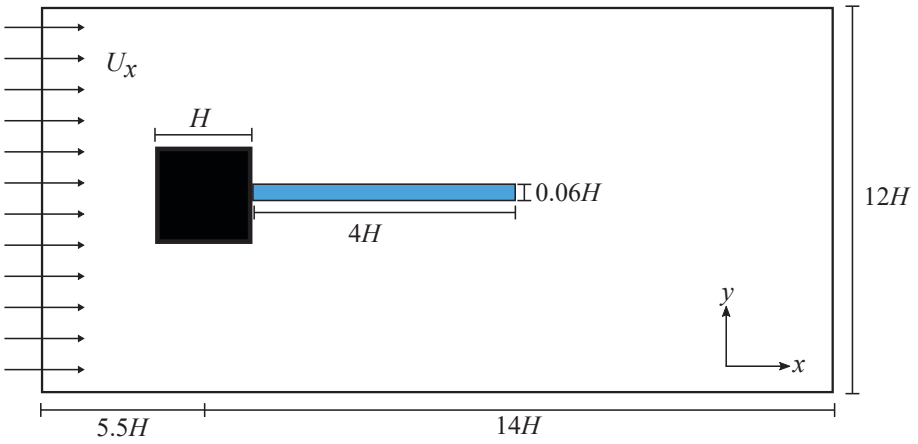

Figure 8: Flexible cantilever attached to a rigid square cylinder: geometry of the computational domain.

This interaction is numerically reproduced by coupling SU2 and Metafor using CUPyDO. A nonlinear formulation for the structural part is necessary to

Solid	Density [$\mathrm{kg\,m^{-3}}$]	ρ_s	100
	Young's modulus [Pa]	E	$2.5 \cdot 10^5$
	Poisson's ratio [-]	ν_s	0.35
Fluid	Density [$\mathrm{kg\,m^{-3}}$]	ρ_f	1.18
	Kinematic viscosity [$\mathrm{m^2\,s^{-1}}$]	ν_f	$1.54 \cdot 10^{-5}$

Table 1: Flexible cantilever attached to a rigid square cylinder: physical properties of the solid and fluid.

correctly predict the bending of the cantilever undergoing large displacements. The fluid domain is solved using the compressible laminar Navier-Stokes equations on a hybrid structured-unstructured grid with 15798 cells. The mesh is globally unstructured with a structured layer near the solid boundary. The cantilever is modeled as pure elastic material and discretized with 240 $\times$ 10 (length $\times$ thickness) quad elements. In order to eliminate any interpolation error, discretization is performed so as to have matching meshes at the fluid-structure interface.

The time step for the simulation is $\Delta t = 0.0025$ s, which corresponds to 122 time steps per period for the first bending mode of the beam. Four BGS iterations with no relaxation are typically required to reach a coupling tolerance of 10^{-6} m (10^{-4} times the expected tip displacement) on the structural displacement. The simulation starts with uniform flow and no initial displacement of the cantilever. Figure 9 shows the computed tip displacement d_z as a function of time. At the start of the simulation, a transient behavior is observed until the vortex shedding, and consequently the tip displacement, reaches an established regime where the displacement amplitude is clearly modulated by higher frequency waves. This stems from the complex structure of the vortex shedding, as was already observed by Sanchez *et al.* [63] who also used the SU2 solver but with a native FSI implementation.

A summary of results from the literature (e.g., [64, 48, 65]) is provided by Habchi *et al.* [49]. The oscillation frequency typically falls in the range $2.94 - 3.25$ Hz, while the amplitude of the tip displacement is in the range $0.95 - 1.15$ cm, as summarized in Table 2. The present computation predicts

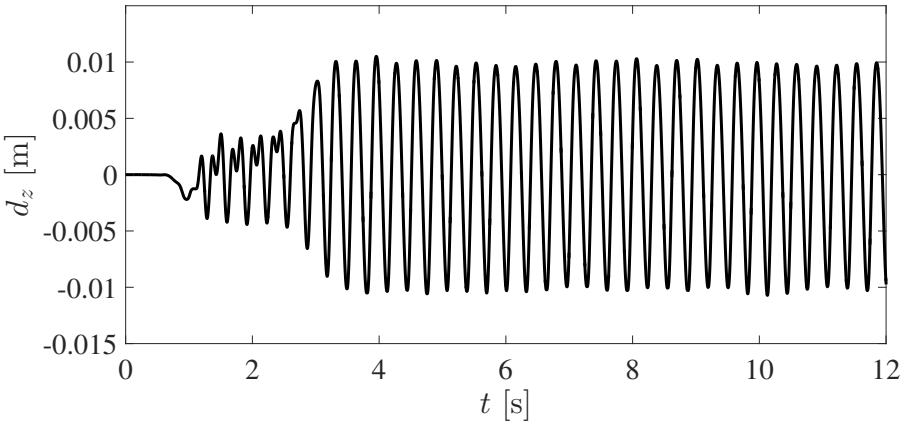

Figure 9: Displacement of the flexible cantilever tip as a function of time.

a maximum tip displacement $d_y = 1.07$ cm and a frequency $f = 3.14$ Hz , which is in very good agreement with results from the literature. Figure 10

	d_y (cm)	f (1/s)
CUPyDO	1.07	3.14
Sanchez *et al.* [63]	1.05-1.15	3.05-3.15
Habchi *et al.* [49]	1.02	3.25
Kassiotis *et al.* [64]	1.05	2.98
Wood *et al.* [48]	1.15	2.94
Olivier *et al.* [65]	0.95	3.17

Table 2: Comparison of the maximum tip displacement and oscillation frequency of the flexible cantilever between the present computation and results from the literature. The range of values obtained by Sanchez *et al.* corresponds to a parametric study on the relaxation parameter in the BGS algorithm.

shows the velocity magnitude contour at several time steps of a period T, where the vortical flow structures and the large displacement of the cantilever can be observed.

4.3. Aeroelastic study of the AGARD 445.6 wing

The experimental AGARD 445.6 wing test case [66] is a frequently used three-dimensional validation case for transonic flutter simulations. The present computational study is based on the weakened model 3 of the wing. This is a

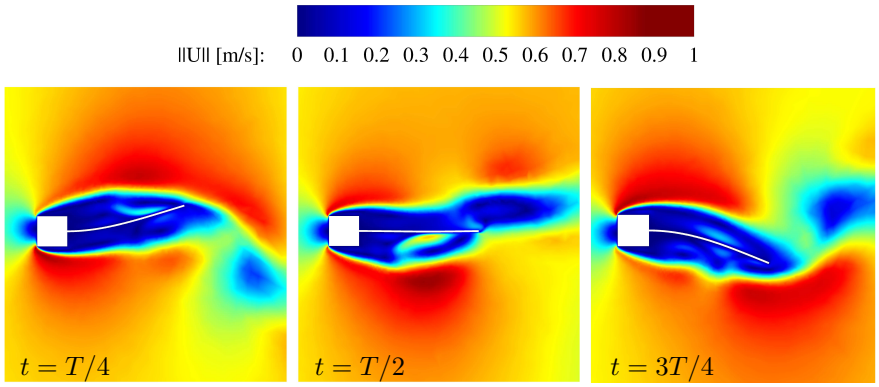

Figure 10: Flexible cantilever attached to a rigid square cylinder: velocity magnitude contour at three phases of a period.

45° swept-back wing whose geometrical properties are summarized in Table 3. The cross section is a symmetric NACA 65a004 airfoil and the wing is clamped at the root.

Root chord [m]	c_r	0.559
Taper ratio [-]	λ	0.658
Tip chord [m]	c_t	0.368
Semi-span [m]	b_s	0.762
Aspect ratio [-]	AR	1.644
Wing surface [m^2]	S	0.353
Mean aerodynamic chord [m]	$\bar{c}$	0.470

Table 3: Geometrical properties of the AGARD 445.6 wing.

The solid wing is modeled in Metafor with 8-node continuum elements and an orthotropic elastic material whose properties are summarized in Table 4. It is discretized with 31, 17 and 2 cells in the spanwise, chordwise and thickness direction, respectively. A modal analysis is first performed and the first four natural frequencies computed are compared with results in the literature in Table 5, showing good agreement with models coming from other references.

The fluid part of the problem is discretized using a structured O-mesh with

Longitudinal Young's modulus [GPa]	E_1	3.151
Transverse Young's moduli [GPa]	E_2, E_3	0.4162
Shear moduli [GPa]	G_{12}, G_{13}, G_{23}	0.4392
Poisson's ratio [-]	$\nu_{12}, \nu_{13}, \nu_{23}$	0.31
Density [$\mathrm{kg\,m^{-3}}$]	ρ_s	381.98

Table 4: Material properties for the AGARD 445.6 wing.

	f_1	f_2	f_3	f_4
Metafor	9.54	40.35	50.22	97.67
Yates [66]	9.60	38.10	50.70	98.50
Goura [67]	9.67	36.87	50.26	90.00
Beaubien *et al.* [68]	9.46	39.44	49.71	94.39
Zhang*et al.* [69]	9.57	38.17	48.35	91.55

Table 5: First four natural frequencies of the AGARD 445.6 wing from the present calculation and the literature. Frequencies are in Hz with f_1 and f_3 corresponding to the first and second bending modes, and f_2 and f_4 to the first and second torsion modes, respectively.

a total number of 248000 cells. The fluid domain extends up to $25c_r$ from the wing in each direction. The wing surface is discretized with 30, 50 and 20 cells in the spanwise, chordwise and thickness direction respectively. The mesh around the wing is illustrated in Figure 11.

A coupled simulation is used to compute the flutter boundary of the wing. The compressible solver of SU2 is used to solve the Euler equations for the fluid part of the problem. A symmetry boundary condition is imposed on the plane in which the wing is clamped. Similarly to the experimental investigation, a large range of Mach numbers, from $\mathrm{M}_\infty = 0.499$ to $\mathrm{M}_\infty = 1.141$, is simulated. Based on experimental conditions, the corresponding Reynolds numbers are in the range $\mathrm{Re} = 0.54 \cdot 10^6$ to $\mathrm{Re} = 1.89 \cdot 10^6$. As for the Isogai wing section test case, computations are performed with variable speed indices for a given Mach number and flutter is inferred from the damping coefficients extracted from the aeroelastic response. The speed index for the AGARD 445.6 wing test case is defined as

$$V^* = \frac{U_\infty}{0.5 c_r \omega_2 \sqrt{\mu}}\,, \tag{41}$$

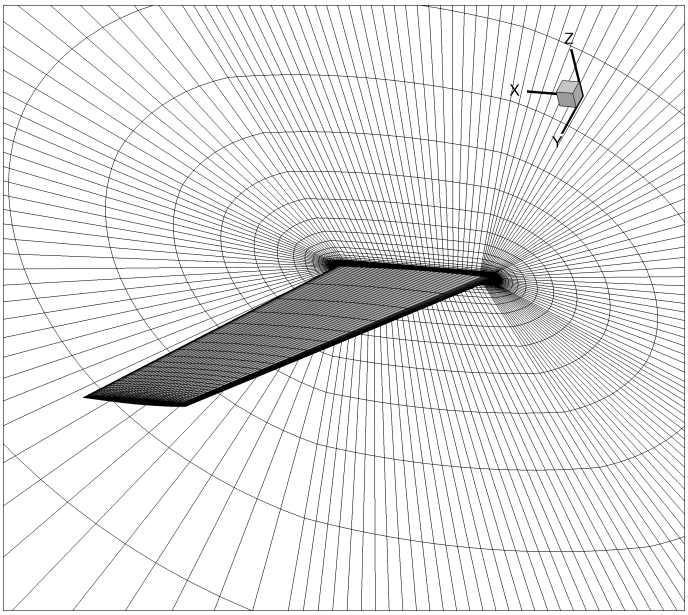

Figure 11: CFD mesh around the AGARD 445.6 wing.

where c_r, ω_2 and μ are the root chord, the first torsion natural frequency and the mass ratio, respectively. The mass ratio is given by $\mu = m/\rho_\infty V$ where $m = 1.863$ kg and $V = 0.130$ m^3 are taken from Yates [66].

Since the discretization of the two interfaces is not matching, the mesh interpolator of the coupling tool is used to map the two interface meshes and to communicate the data. As the number of points on the interface is limited, TPS interpolation can be used for high accuracy without drastically impacting the computational cost. The simulation is performed with a time step of 0.001 s ($\approx$ 105 time steps per period of the first bending mode) with no relaxation on the BGS coupling. In order to limit the computation time of the simulation, a maximum number of four coupling iterations is imposed which is typically sufficient to reach a coupling tolerance of order 10^{-7} m (about 10^{-5} times the expected displacement amplitude at flutter).

The simulation is initialized with uniform flow and no deformation of the solid wing. During the first 0.01 s of simulation, a vertical load is applied on an upstream portion of the wing tip in order to induce a small perturbation

of a determined amplitude (around 0.66 % of the span). Then the loading is released and the wing is free to vibrate in the flow. The aeroelastic response is illustrated in Figure 12, which shows the vertical displacement of the leading edge at the wing tip for three different speed indices around the flutter boundary at $M_\infty = 0.96$. Simulations are performed in parallel on 16 cores (16 fluid instances, 1 solid) of a computing node with Intel Xeon E5-2650 processor (2 GHz, 16 threads).

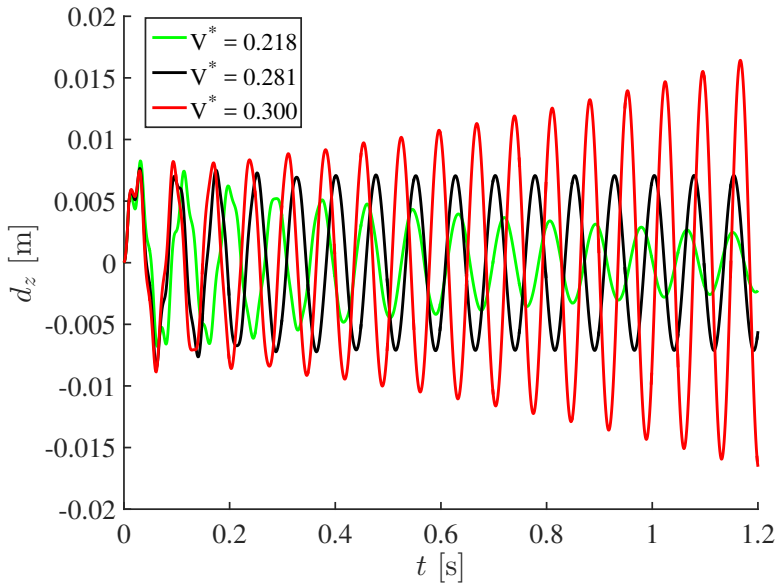

Figure 12: Aeroelastic response of the AGARD 445.6 for three speed indices V^* with $M_\infty = 0.96$ and $\rho_\infty = 0.0634$ kg/m^3: vertical displacement d_z of the leading edge at the wing tip.

As previously mentioned, the flutter condition is determined by successive evaluations of the damping coefficient ζ as a function of the speed index. Figure 13 illustrates the standard evolution of the damping coefficient as a function of the speed index at $M_\infty = 0.96$. It first increases starting from low values at low speed indices, then reaches a maximum value and finally drops until it crosses the $\zeta = 0$ axis corresponding to the flutter point.

Figure 14 shows the pressure contours on the wing at $M_\infty = 0.96$ and $V^* = 0.300$ for three time steps over a cycle. Contours of the Mach number in the

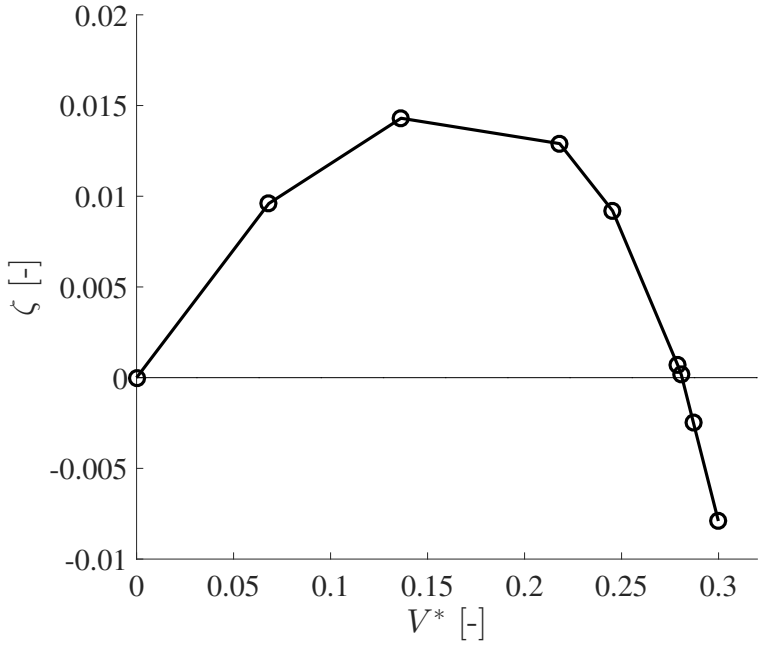

Figure 13: Evolution of the computed damping coefficient of the AGARD 445.6 wing aeroelastic response as a function of the speed index for $M_\infty = 0.96$ and $\rho_\infty = 0.0634$ kg/m^3.

supersonic region are superposed at three wing sections in order to show its motion within one cycle and thus highlight the non-linearities typical of the transonic regime.

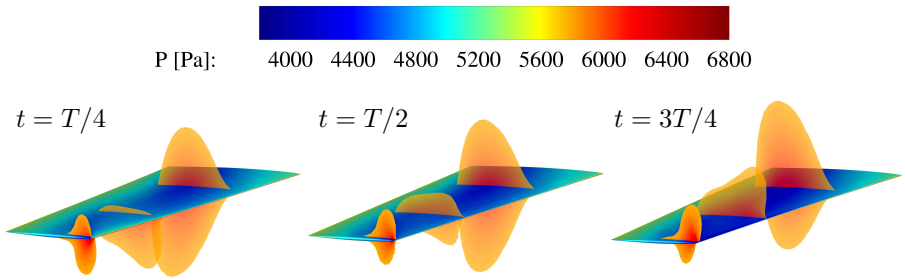

Figure 14: Surface pressure and Mach number in supersonic region for the AGARD 445.6 wing at three different times of a period with $M_\infty = 0.96$, $\rho_\infty = 0.0634$ kg/m^3 and $V^* = 0.300$.

Figure 15 shows the computed flutter boundary that is compared to the experimental results [66] and to computational results obtained solving the Euler [68, 70, 59, 71] or RANS [3, 71] equations. For all the computational reference results, the structural part is modelled by a modal decomposition approach which differs from the fully time-integrated approach used in this paper. It can be seen that the results obtained by coupling SU2 and Metafor with CUPyDO are in good agreement with experiments and with the other computational results found in the literature in the subsonic and transonic regimes. It is also important to note that the transonic dip is well-captured. However, larger discrepancies with the experimental data are observed for supersonic Mach numbers, especially at $M_\infty = 1.141$. Similarly to other computations, the flutter boundary is over-estimated at these Mach numbers. Although the origin of this discrepancy remains unclear, several explanations [59, 71] have been proposed, such as the impact of viscous effects (not accounted for in the present Euler simulations) and the complex nonlinear shock - boundary layer interaction. Even for RANS simulations, uncertainties remain concerning the turbulence model and the impact of transition. Additionally, the wing tip geometry, i.e., cut-off or rounded (rounded here), and spatial discretization scheme (centered or upwind) [3] could also play a role in the supersonic regime. Finally, the effect of structural damping on the flutter boundary should be investigated, since no indication of its experimental value is given [70]. All these details are relevant for accurately capturing the flutter boundary at high Mach numbers but are beyond the scope of this paper.

4.4. Flexible plate in a cross flow

A test case involving stronger added-mass effects is considered in order to asses the performance of coupling relaxation. A flexible linear elastic plate is immersed in a cross flow and clamped at the bottom so that it bends under its own drag. This case is directly taken from Tian *et al.* [7] and originally comes from an experimental study made by Luhar and Nepf [72] on the deformation of aquatic plants submitted to a water flow. The computational domain as well

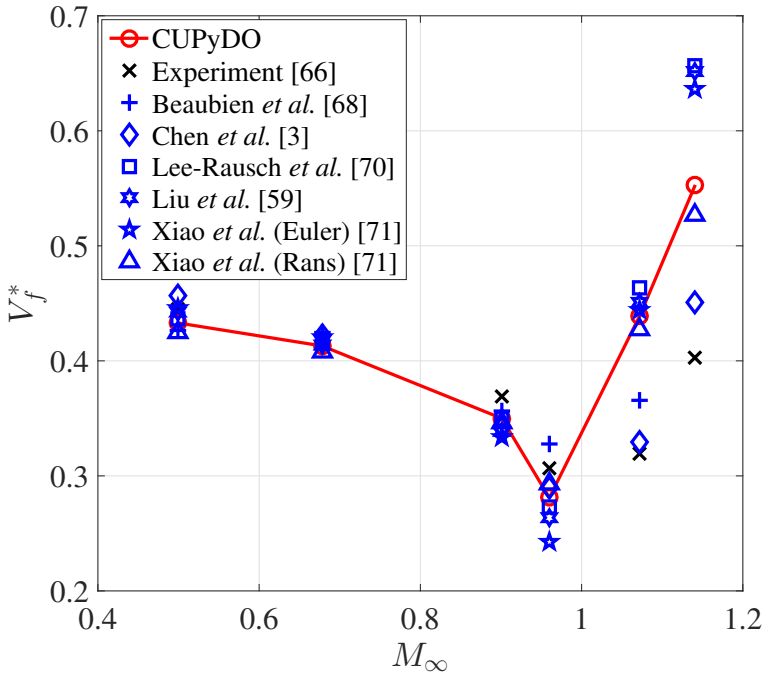

Figure 15: Flutter boundary of the AGARD 445.6 wing in the transonic regime.

as the fluid boundary conditions are represented in Figure 16. A flexible plate of length L, width b and thickness h is placed vertically in a uniform flow of velocity U_∞, density ρ_∞ and viscosity μ_∞. One side of the plate is clamped and all the other sides are free. The external boundary of the domain is a rectangular box extending from $(-5b, -8b, -8.5b)$ to $(16b, 8b, 8.5b)$. The plate is fixed at the origin of the xy-plane. The dimensions of the plate are such that $L = 5b$ and $h = 0.2b$. In this case a value of 0.1 m for the plate width b is chosen. Note that the plane in which the plate is clamped is considered as an inviscid wall to neglect the boundary layer and conserve upstream flow uniformity. The following non-dimensional parameters are used: $Re = U_\infty b/\nu_\infty = 1600$ where ν_∞ is the fluid kinematic viscosity, $E^* = E/\rho_\infty U_\infty^2$ where E is the Young's modulus of the plate, and $\rho^* = \rho_s/\rho_\infty = 0.678$ where ρ_s is the plate density. The Poisson's ratio is set to $\nu_s = 0.4$.

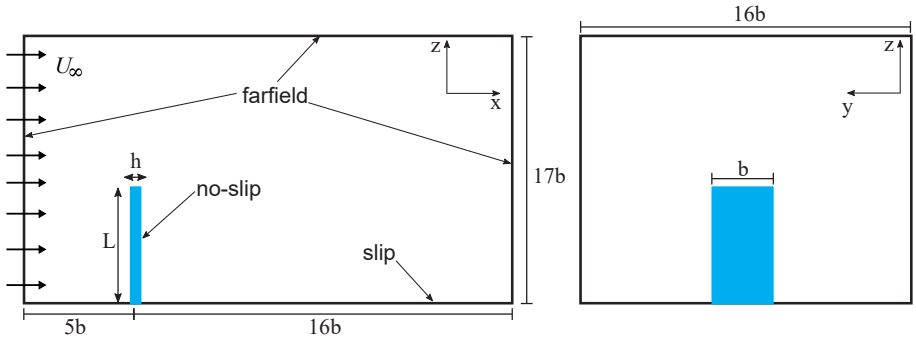

Figure 16: Plate in a cross flow: geometry and computational domain and fluid boundary conditions.

The fluid domain is discretized into a structured mesh with a total number of 193900 cells. The plate is discretized with 40, 15 and 10 cells along the length, width and thickness, respectively. The structural domain is discretized with a structured mesh with 8-node continuum elements so that the meshes are matching at the fluid-structure interface. The fluid is solved using the laminar compressible Navier-Stokes equations in SU2 while the structural displacements are computed with Metafor. Although the solution obtained by Tian *et al.* [7] seems to be stationary, the simulation with CUPyDO is performed with unsteady time integration in order to capture the complex transient part of the response of the plate and assess the performance of the relaxation scheme of the BGS coupling algorithm during the whole time integration. The time step of the simulation is set to 0.005 s which corresponds approximately to 1/400 of the time needed to reach a steady-state solution. Because the fluid and solid densities have the same order of magnitude, making the coupled system converge is challenging. In particular, only low values of the static relaxation parameter (i.e., below 0.5) lead to a stable solution, but at the expense of a very low convergence rate. A large number of coupling iterations (up to 87) is needed to reach the specified coupling tolerance of 10^{-6} m ($4 \cdot 10^{-6}$ times the displacement of the plate along the x-axis).

The dynamic Aitken relaxation strategy is thus used to achieve convergence while keeping the number of iterations per time step at an acceptable level. A

minimal criterion (see Section 3.3) is used to restrict the relaxation parameter to a low value of 0.1 at the beginning of each time step, but this value naturally increases along the coupling iterations allowing the iterative procedure to reach the desired tolerance within 6 to 20 iterations. The computation is run on 16 cores (16 fluid instances, 1 solid) of the same computing node used for the AGARD 445.6 test case and needs about 2 seconds of simulated time for the plate to reach a steady state deflection. Figure 17 shows the displacement of the tip face's center of the plate as a function of time. At the beginning of the simulation, the plate is rapidly bent under the effect of drag and then a small restoration is observed before reaching a steady-state deformation, which is depicted in Figure 18 where streamlines are superposed in order to highlight the vorticity dynamics in the near field behind the plate. Note that no vortex shedding is observed due to three-dimensional effects for which the tip vortices significantly influence the vortex dynamics and tend to smooth out the unsteadiness that typically appears in two-dimensional simulations. This behavior was already highlighted with flow simulations around rigid flat plate wings featuring comparable aspect ratios but lower Reynolds numbers than in the present case [73]. Table 6 gives the comparison between the present results and those

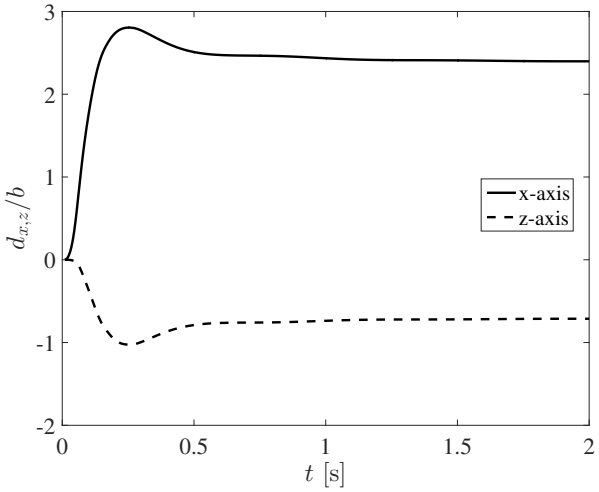

Figure 17: Displacements along the x and z-axis of the plate tip as a function of time.

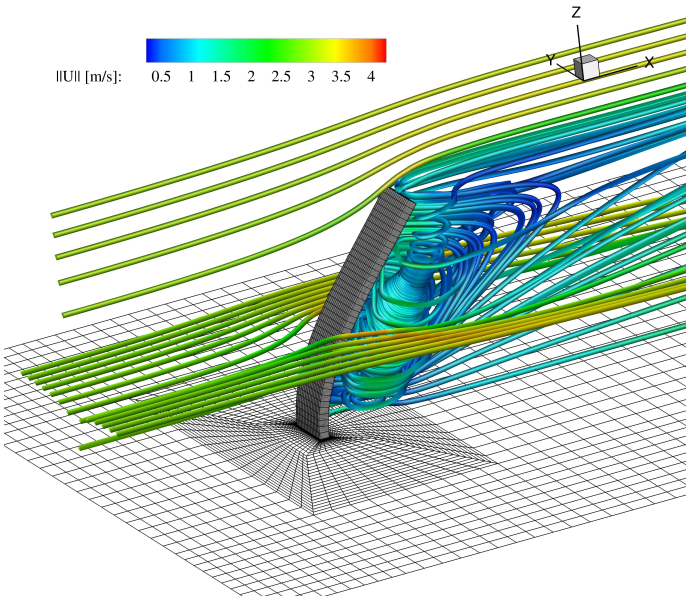

Figure 18: Steady-state deformation of the plate in a cross flow with steamlines colored with velocity magnitude.

obtained by Tian *et al.* [7]. Similar values are obtained with a maximum discrepancy of 13.8% on the drag coefficient and a minimal discrepancy of 1.6% for the displacement along the x-axis. Although the root causes for these discrepancies could not be unequivocally identified, the results of several tests suggest that they originate in the fluid solution. In particular, they likely stem from the present use of a compressible solver, while Tian *et al.* [7] results are based on an incompressible approach. Additionally, the respective numerical methods also differ (immersed boundary [7] vs. ALE with mesh deformation). Nonetheless, this test case demonstrates the performance of the Aitken relaxation implemented in the coupling algorithm of CUPyDO.

	C_D	d_x/b	d_z/b
CUPyDO	1.07	2.41	-0.72
Tian *et al.* [7]	0.94	2.45	-0.75

Table 6: Comparison of the drag coefficient and tip displacements of the plate between the present computation and the results of Tian *et al.* [7].

4.5. CHT with a heated hollow cylinder

The last validation test case involves the thermal coupling capability of CUPyDO. The test case is taken from Nettis [51]. This is a perfectly rigid hollow cylinder immersed in a uniform flow, as illustrated in Figure 19. The ratio between the inner and outer diameter is $D_c/D = 0.5$. A temperature $T_c = 350$ K is imposed on the inner boundary of the cylinder. The farfield fluid temperature is also imposed, $T_\infty = 288.15$ K, so that the thermal exchange at the outer boundary of the cylinder defines a convective CHT problem. Nettis [51] performed the simulation using a compressible fluid solver with the parameters Re = 40, Pr = 0.72 and $M_\infty = 0.38$ for the Reynolds, Prandtl and Mach numbers, respectively. The ratio between the thermal conductivities λ_s/λ_f is set to 4. The coupled simulation is here reproduced with the same parameters using SU2 to solve the fluid problem and GetDP to solve the thermal conduction problem within the solid. A steady coupled simulation is used since the Reynolds number is low enough to guarantee a stationary flow in the wake of the cylinder. The

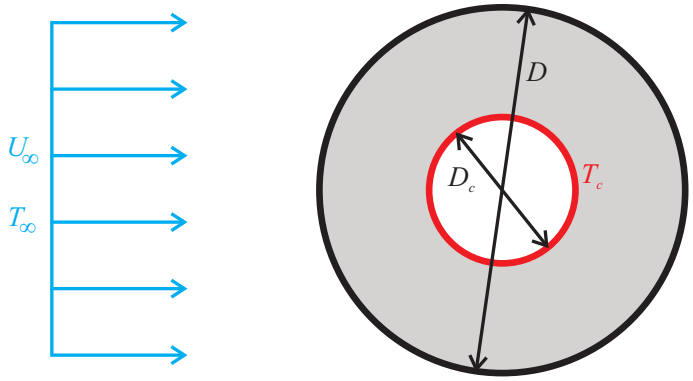

Figure 19: Heated hollow cylinder on a cross flow.

fluid domain is circular and extends up to $25D$. It is discretized by a structured O-mesh with 50 cells in the radial direction and 120 cells in the circumferential direction. The structural domain is discretized into a structured mesh with 20 quad elements in the radial direction and the meshes are matching at the fluid-structure interface.

The four thermal coupling schemes presented in Section 3.4 are tested. Only the TFFB scheme is unstable since the Biot number is below 1 for this particular case. The three other schemes provide a converged solution. In the case of the hFFB scheme, the value of the numerical heat transfer coefficient influences the number of iterations needed to reach a tolerance of 10 W/m^2 on the heat flux at the interface. This is summarized in Table 7, showing that increasing the value of h_c reduces the number of coupling iterations. For tested values higher than $h_c = 20$, the coupled simulations become unstable. The hFTB scheme is less sensitive to the value of h_c. As shown in Table 7, the same number of iterations is necessary to reach the specified temperature tolerance of 0.1 K on the interface (leading to an equivalent level of convergence as for the hFFB scheme). For values higher than $h_c = 1$ the scheme is unstable. It is also shown that the FFTB scheme takes the same number of iterations as hFTB to reach the same tolerance.

	Value of h_c (W/m^{2}K)	Coupling iterations
hFFB	1	198
	5	52
	10	32
	15	23
	20	19
hFTB	0.01	8
	0.05	8
	0.1	8
	0.5	8
	1	8
FFTB	-	8

Table 7: Comparison of the performances for different CHT coupling schemes.

Figure 20 shows the temperature distribution at the fluid-structure interface T_w obtained with the hFTB and hFFB couplings in CUPyDO. Good agreement is obtained between the present hFTB calculations and the results from Nettis [51] where a hFTB scheme and a compressible solver to compute the fluid part of the problem are used as well. Discrepancies between hFTB and hFFB

are observed mainly on the upstream part of the cylinder surface, however the temperature difference at the stagnation point between the two schemes is not higher than 0.2%. The temperature field inside the solid domain is illustrated in Figure 21 showing the non-symmetric distribution due to the convective heat transfer induced by the surrounding flow.

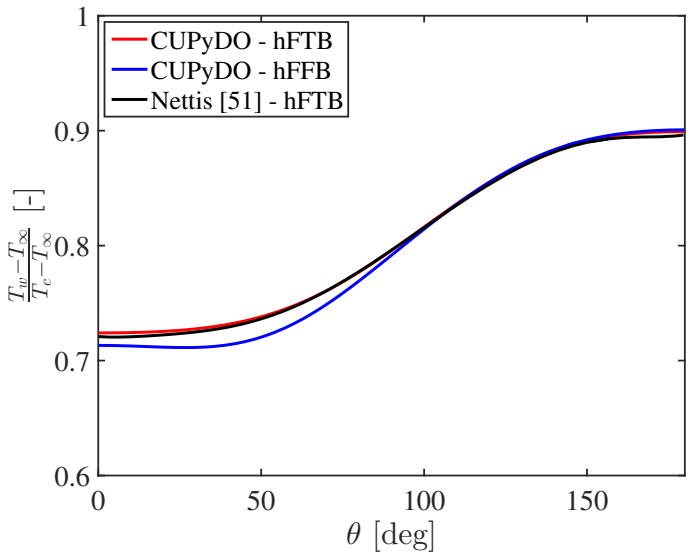

Figure 20: Temperature distribution on the wetted surface of the cylinder. The upstream stagnation point corresponds to 0°.

5. CONCLUSION AND FUTURE WORK

CUPyDO, a modular and flexible implementation of a coupling environment for fluid-structure interaction problems has been presented. The coupled problem is solved using a partitioned approach in which the fluid and solid solvers are integrated in a single coupling environment and communicate through a Python wrapping layer. This ensures that the high-level management of the two solvers (black-box tools) is very intuitive and flexible and that all the intensive calculations remain embedded in their respective core codes. The object-oriented architecture of CUPyDO is designed to guarantee the compatibility of

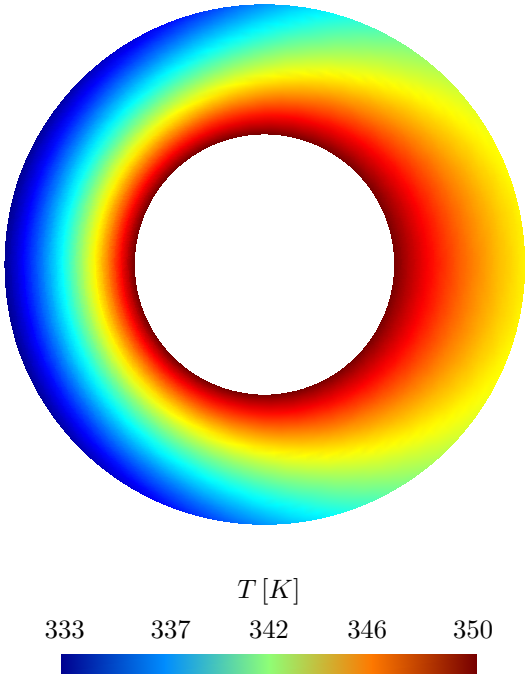

Figure 21: Temperature distribution inside the hollow cylinder.

each coupled solver with minimal adaptation effort by using dedicated interfacing classes based on generic fluid and solid classes. The coupling tool is designed to work on parallel environments using the MPI protocol so that engineering studies involving large systems can be considered. Communications between the solvers, data interpolations and coupling algorithms are parallelized in order to avoid any bottleneck due to partial serialization of several coupling procedures. Parallel algebraic operations are supported by the PETSc library that is used in CUPyDO with dedicated Python bindings. Interpolation based on Radial Basis Functions is used to transfer data across the fluid-structure interface when the two meshes are not matching. The time-marched block Gauss-Seidel algorithm is used to strongly couple the two domains and Aitken under-relaxation is used to stabilize the coupling when significant added-mass effects are involved.

The results of several test cases have been compared to the literature, demonstrating the accuracy of the coupling tool for coupled problems of various complexity. To compute the fluid part of the problem, the Euler or Navier-Stokes equations are solved in SU2 with the finite volume method in a Arbitrary Lagrangian-Eulerian formulation. The high modularity of the framework has been demonstrated by using different structural solvers and models. Lagrangian linear/non-linear finite element solvers such as GetDP or Metafor are used to solve the structural dynamic equilibrium for deformable solids and the heat equation for thermal conduction. A simpler integrator is used to compute constrained rigid body motions as in the Isogai aeroelastic test case.

Future work will focus on extending the current capabilities of the tool. In particular, this includes the implementation of a Newton method for the coupling algorithm and other interpolation methods. The parallel scalability of the implementation should also be assessed in order to identify potential improvements to the parallelization. Another avenue for future work is to interface other fluid and solid solvers with CUPyDO, as it has already been done with a PFEM solver. This would also provide the opportunity to tackle other multiphysics applications, for which the technical implementation in CUPyDO would simply require a straightforward extension of the quantities that are transferred between the solvers (e.g., species flux for problems with chemical reactions, such as in ablation). The main challenge in this case would rather stem from possible numerical instabilities and convergence issues of the coupling algorithm. Other multiphysics problems might also require the coupling of more than two different solvers. Such a development could be envisaged, but would be challenging due to the higher complexity of the resulting communication network and coupling algorithm.

ACKNOWLEDGEMENT

The authors would like to acknowledge the Walloon Region and Walloon-Brussels Federation as research funding entities of this project under Grant No. 7093, the *Consortium des Equipements de Calculs Intensifs* (CÉCI), funded

by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11, for providing the computational resources required for the three dimensional test cases presented in Sections 4.3 and 4.4 and Professor Christophe Geuzaine (University of Liège) for his collaborative work on the interface between GetDP and CUPyDO.

REFERENCES

[1] V. Kalro and T.E. Tezduyar. A parallel 3D computational method for fluid-structure interactions in parachute systems. *Computer Methods in Applied Mechanics and Engineering*, 190:321–332, 2000. https://doi.org/10.1016/S0045-7825(00)00204-8.

[2] R. Kamakoti and W. Shyy. Fluid-structure interaction for aeroelastic applications. *Progress in Aerospace Sciences*, 40(8):535–558, 2004. https://doi.org/10.1016/j.paerosci.2005.01.001.

[3] X. Chen, G-C. Zha, and M-T. Yang. Numerical simulation of 3-D wing flutter with fully coupled fluid-structure interaction. *Computers and Fluids*, 36(5):856–867, 2007. https://doi.org/10.1016/j.compfluid.2006.08.005.

[4] J.R.R.A. Martin, J.J. Alonso, and J.J. Reuther. High-fidelity aerostructural design optimization of a supersonic business jet. *Journal of Aircraft*, 41(3): 523–530, 2004. https://doi.org/10.2514/1.11478.

[5] H. Luo, R. Mittal, X. Zheng, S.A. Bielamowicz, R.J. Walsh, and J.K. Hahn. An immersed-boundary method for flow-structure interaction in biological systems with application to phonation. *Journal of Computational Physics*, 227(22):9303–9332, 2008. https://doi.org/10.1016/j.jcp.2008.05.001.

[6] Y. Wu and X-C. Cai. A fully implicit domain decomposition based ALE framework for three-dimensional fluid-structure interaction with applica-

tion in blood flow computation. *Journal of Computational Physics*, 258: 524–537, 2014. https://doi.org/10.1016/j.jcp.2013.10.046.

[7] F-B. Tian, H. Dai, H. Luo, J.F. Doyle, and B. Rousseau. Fluid-structure interaction involving large deformations: 3D simulations and applications to biological systems. *Journal of Computational Physics*, 258:451–469, 2014. https://doi.org/10.1016/j.jcp.2013.10.047.

[8] R. Wüchner, A. Kupzok, and K-U. Bletzinger. A framework for stabilized partitioned analysis of thin membrane-wind interaction. *International Journal for Numerical Methods in Fluids*, 54:945–963, 2007. https://doi.org/10.1002/fld.1474.

[9] G. Hou, J. Wang, and A. Layton. Numerical methods for fluid-structure interaction - A review. *Communications in Computational Physics*, 12(2): 337–377, 2012. https://doi.org/10.4208/cicp.291210.290411s.

[10] L. Garelli. *Fluid-structure interaction using an arbitrary Lagrangian-Eulerian formulation*. PhD thesis, Universidad Nacional Del Litoral, 2011.

[11] S.R. Idelsohn, E. Oñate, R. Rossi, J. Marti, and F. Del Pin. New computational challenges in fluid-structure interactions problems. In J Eberhardsteiner, C Hellmich, HA Mang, and J Périaux, editors, *ECCOMAS Multidisciplinary Jubilee Symposium*, volume 14 of *Computational Methods in Applied Sciences*, pages 17–31. Springer Netherlands, Dordrecht, 2009. https://doi.org/10.1007/978-1-4020-9231-2_2.

[12] H-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, and B. Uekermann. preCICE - A fully parallel library for multi-physics surface coupling. *Computers and Fluids*, 141:250–258, 2016. https://doi.org/10.1016/j.compfluid.2016.04.003.

[13] Fraunhofer Institute for Algorithms and Germany Scientific Computing SCAI, Sankt Augustin. Mpcci 4.5.0-1 documentation. https://www.

mpcci.de/content/dam/scai/mpcci/documents/MpCCIdoc-4_5_0.pdf. Accessed October 31, 2017.

[14] S. Kataoka, S. Minami, H. Kawai, T. Yamada, and S. Yoshimura. A parallel iterative partitioned coupling analysis system for large-scale acoustic fluid-structure interactions. *Computational Mechanics*, 53:1299–1310, 2014. https://doi.org/10.1007/s00466-013-0973-1.

[15] F. Duchaine, S. Jauré, D. Poitou, E. Quémerais, G. Staffelbach, T. Morel, and L. Gicquel. Analysis of high performance conjugate heat transfer with the OpenPALM coupler. *Computational Science and Discovery*, 8, 2015. https://doi.org/10.1088/1749-4699/8/1/015003.

[16] T.D. Economon, F. Palacios, S.R. Copeland, T.W. Lukaczyk, and J.J. Alonso. SU2: An open-source suite for multiphysics simulation and design. *AIAA Journal*, 54(3):828–846, 2016. https://doi.org/10.2514/1.J053813.

[17] F.R. Menter. Zonal two equation $k-\omega$ turbulence model for aerodynamic flows. In *AIAA Paper 1993-2906. 23rd Fluid Dynamics, Plasmadynamics, and Lasers Conference*, Orlando, Florida, USA, 1993. https://doi.org/10.2514/6.1993-2906.

[18] P.R. Spalart and S.R. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, 1992. https://doi.org/10.2514/6.1992-439.

[19] J. Donéa, A. Huerta, J-Ph. Ponthot, and A. Rodríguez-Ferran. Arbitrary lagrangian-eulerian methods. In R Stein, R de Borst, and TJR Hughes, editors, *Encyclopedia of Computational Mechanics*. Wiley, 2004. https://doi.org/10.1002/0470091355, ISBN 9780470091357.

[20] F. Palacios, M.R. Colonno, A.C. Aranake, A. Campos, S.R. Copeland, T.D. Economon, A.K. Lonkar, T.W. Lukaczyk, T.W.R. Taylor, and J.J. Alonso.

Stanford University Unstructured (SU^2): An open-source integrated computational environment for multi-physics simulation and design. In *AIAA Paper 2013-0287. 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition 2013*, Grapevine, Texas, USA, 7-10 January, 2013. https://doi.org/10.2514/6.2013-287.

[21] T.D. Economon, D. Mudigere, G. Bansal, A. Heinecke, F. Palacios, J. Park, M. Smelyanskiy, J.J. Alonso, and P. Dubey. Performance optimizations for scalable implicit RANS calculations with SU2. *Computers and Fluids*, 129: 146–158, 2016. https://doi.org/10.1016/j.compfluid.2016.02.003.

[22] R. Sanchez, H.L. Kline, D. Thomas, A. Variyar, M. Righi, T.D. Economon, J.J. Alonso, R. Palacios, G. Dimitriadis, and V. Terrapon. Assessment of the fluid-structure interaction capabilities for aeronautical applications of the open-source solver SU2. In *ECCOMAS Congress, VII European Congress on Computational Methods in Applied Sciences and Engineering*, Crete Island, Greece, June 2016. https://doi.org/10.7712/100016.1903.6597.

[23] M. Pini, S. Vitale, P. Colonna, G. Gori, A. Guardone, T.D. Economon, J.J. Alonso, and F. Palacios. SU2: the open-source software for non-ideal compressible flows. *Journal of Phyics: Conference Series*, 821(1), 2017. https://doi.org/10.1088/1742-6596/821/1/012013.

[24] A. Jameson and S. Schenectady. An assessment of dual-time stepping, time spectral and artificial compressibility based numerical algorithms for unsteady flow with applications to flapping wings. In *AIAA Paper 2009-4273. 19th AIAA Computational Fluid Dynamics Conference*, San Antonio, Texas, USA, 2009. https://doi.org/10.2514/6.2009-4273.

[25] J.T. Batina. Unsteady Euler algorithm with unstructured dynamic mesh for complex-aircraft aerodynamic analysis. *AIAA Journal*, 29(3):327–333, 1991. https://doi.org/10.2514/3.10583.

[26] C. Farhat, C. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer Methods in Applied Mechanics and Engineering*, 163:231–245, 1998. https://doi.org/10.1016/S0045-7825(98)00016-4.

[27] R. Löhner and C. Yang. Improved ALE mesh velocities for moving bodies. *Communications in Numerical Methods in Engineering*, 12:599–608, 1996. https://doi.org/10.1002/(SICI)1099-0887(199610)12:10<599::AID-CNM1>3.0.CO;2-Q.

[28] M. Lesoinne and C. Farhat. Geometric conservation laws for flow problems with moving boundaries and deformable meshes, and their impact on aeroelastic computations. *Computer Methods in Applied Mechanics and Engineering*, 134:71–90, 1996. https://doi.org/10.1016/0045-7825(96)01028-6.

[29] T.D. Economon, F. Palacios, and J.J. Alonso. Unsteady continuous adjoint approach for aerodynamic design on dynamic meshes. *AIAA Journal*, 53 (9):2437–2453, 2015. https://doi.org/10.2514/1.J053763.

[30] M.L. Cerquaglia, G. Deliège, R. Boman, L. Papeleux, and J.P. Ponthot. The particle finite element method for the numerical simulation of bird strike. *International Journal of Impact Engineering*, 109:1 – 13, 2017. https://doi.org/10.1016/j.ijimpeng.2017.05.014.

[31] E. Oñate, S.R. Idelsohn, F. Del Pin, and R. Aubry. The particle finite element method - An overview. *International Journal of Computational Methods*, 01(02):267–307, 2004. https://doi.org/10.1142/S0219876204000204.

[32] METAFOR. A nonlinear finite element code. University of Liège, http://metafor.ltas.ulg.ac.be/. Accessed October 31, 2017.

[33] Y. Crutzen, R. Boman, L. Papeleux, and J-P. Ponthot. Continuous roll forming including in-line welding and post-cut within an ALE formalism.

Finite Elements in Analysis and Design, 143:11–31, 2018. https://doi.org/10.1016/j.finel.2018.01.005.

[34] J-P. Ponthot. Unified stress update algorithms for the numerical simulation of large deformation elasto-plastic and elasto-viscoplastic processes. *International Journal of Plasticity*, 18(1):91–126, 2002. https://doi.org/10.1016/S0749-6419(00)00097-8.

[35] L. Adam and J-P. Ponthot. Thermomechanical modeling of metals at finite strains: First and mixed order finite elements. *International Journal of Solids and Structures*, 42:5615–5655, 2005. https://doi.org/10.1016/j.ijsolstr.2005.03.020.

[36] P. Bussetta, D. Marceau, and J-P. Ponthot. The adapted augmented Lagrangian method: a new method for the resolution of the mechanical frictional contact problem. *Computational Mechanics*, 49(2):259–275, 2012. https://doi.org/10.1007/s00466-011-0644-z.

[37] R. Boman and J-P. Ponthot. Efficient ALE mesh management for 3D quasi-Eulerian problems. *International Journal For Numerical Methods in Engineering*, 92:857–890, 2012. https://doi.org/10.1002/nme.4361.

[38] C. Geuzaine. GetDP: a general finite-element solver for the de Rham complex. In *PAMM Volume 7 Issue 1. Special Issue: Sixth International Congress on Industrial Applied Mathematics (ICIAM07) and GAMM Annual Meeting, Zürich*, pages 1010603–1010604, 2007. https://doi.org/10.1002/pamm.200700750.

[39] P. Dular, C. Geuzaine, F. Henrotte, and W. Legros. A general environment for the treatment of discrete problems and its application to the finite element method. *IEEE Transactions on Magnetics*, 34(5):3395–3398, 1998. https://doi.org/10.1109/20.717799.

[40] J. Degroote, A. Souto-Iglesias, W. Van Paepegem, S. Annerel, P. Bruggeman, and J. Vierendeels. Partitioned simulation of the interaction between

an elastic structure and free surface flow. *Computer Methods in Applied Mechanics and Engineering*, 199:2085–2098, 2010. https://doi.org/10.1016/j.cma.2010.02.019.

[41] U. Küttler and W. Wall. Fixed-point fluid-structure interaction solvers with dynamic relaxation. *Computational Mechanics*, 43:61–72, 2008. https://doi.org/10.1007/s00466-008-0255-5.

[42] M. Lesoinne and C. Farhat. Stability analysis of dynamic meshes for transient aeroelastic computations. In *AIAA Paper 93-3325. 11th AIAA Computational Fluid Dynamics Conference*, pages 309–314, Orlando, Florida, USA, July 6-9 1993. https://doi.org/10.2514/6.1993-3325.

[43] D.M. Beazley. SWIG : An easy to use tool for integrating scripting languages with C and C++. In *4th Tcl/Tk Workshop*, Monterey, CA, USA, July 1996. https://www.usenix.org/legacy/publications/library/proceedings/tcl96/full_papers/beazley/index.html, Accessed October 31, 2017.

[44] S. van der Walt, S.C. Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. https://doi.org/10.1109/MCSE.2011.37.

[45] Abaqus analysis user's guide V6.14, online documentation. SIMULIA, http://abaqus.software.polimi.it/v6.14/books/usb/default.htm, Accessed October 31, 2017.

[46] L. Dalcin, R. Paz, M. Storti, and J. D'Elía. MPI for Python: Performance improvements and MPI-2 extensions. *Journal of Parallel and Distributed Computing*, 68(5):655–662, 2008. https://doi.org/10.1016/j.jpdc.2007.09.005.

[47] L. Dalcin, R. Paz, P. Kler, and A. Cosimo. Parallel distributed compu-

ting using Python. *Advances in Water Resources*, 34(9):1124–1139, 2011. https://doi.org/10.1016/j.advwatres.2011.04.013.

[48] C. Wood, A.J. Gil, O. Hassan, and J. Bonet. Partitioned block-Gauss-Seidel coupling for dynamic fluid-structure interaction. *Computers and Structures*, 88:1367–1382, 2010. https://doi.org/10.1016/j.compstruc.2008.08.005.

[49] C. Habchi, S. Russeil, D. Bougeard, J-L. Harion, T. Lemenand, A. Ghanem, D. Della Valle, and H. Peerhossaini. Partitioned solver for strongly coupled fluid-structure interaction. *Computers and Fluids*, 71:306–319, 2013. https://doi.org/10.1016/j.compfluid.2012.11.004.

[50] T. Verstraete, Z. Alsalihi, and R.A. Van den Braembussche. A conjugate heat transfer method applied to turbomachinery. In *European Conference on Computational Fluid Dynamics*, TU Delft, The Netherlands, 2006. https://repository.tudelft.nl/islandora/object/uuid:cb5eae7b-6322-4fbf-ad7e-ae10d02f6f27?collection=research, Accessed October 31, 2017.

[51] L. Nettis. Conjugate heat transfer: Strategies and applications. PhD thesis, Politecnico Di Bari, 2011.

[52] A. de Boer, A.H. van Zuijlen, and H. Bijl. Review of coupling methods for non-mathcing meshes. *Computational Methods in Applied Mechanics and Engineering*, 196(8):1515–1525, 2007. https://doi.org/10.1016/j.cma.2006.03.017.

[53] A. de Boer, A.H. van Zuijlen, and H. Bijl. Radial basis functions for interface interpolation and mesh deformation. In Barry Koren and Kees Vuik, editors, *Advanced Computational Methods in Science and Engineering*, pages 143–178. Springer, Berlin, Heidelberg, 2010. https://doi.org/10.1007/978-3-642-03344-5_6.

[54] A. Beckert and H. Wendland. Multivariate interpolation for fluid-structure-interaction problems using radial basis functions. *Aerospace Science and Technology*, 5(2):125 – 134, 2001. https://doi.org/10.1016/S1270-9638(00)01087-7.

[55] T.C.S. Rendall and C.B. Allen. Unified fluid-structure interpolation and mesh motion using radial basis functions. *International Journal for Numerical Methods in Engineering*, 74(10):1519–1559, 2008. https://doi.org/10.1002/nme.2219.

[56] K. Isogai. On the transonic-dip mechanism of flutter of a sweptback wing. *AIAA Journal*, 17(7):793–795, 1979. https://doi.org/10.2514/3.61226.

[57] K. Isogai. Transonic-dip mechanism of flutter of a sweptback wing: Part II. *AIAA Journal*, 19(9):1240–1242, 1981. https://doi.org/10.2514/3.7853.

[58] R.L. Bisplinghoff, H. Ashley, and R.L. Halfman. *Aeroelasticity*. Dover Publications, 1996. ISBN 9780486691893.

[59] F. Liu, J. Cai, Y. Zhu, H.M. Tsai, and A.S.F. Wong. Calculation of wing flutter by a coupled fluid-structure method. *Journal of Aircraft*, 38(2): 334–342, 2001. https://doi.org/10.2514/2.2766.

[60] J.J. Alonso and A. Jameson. Fully-implicit time-marching aeroelastic solution. In *AIAA Paper 94-056. 32nd Aerospace Sciences Meeting and Exhibit*, Reno, NV, USA, 10-13 January, 1994. https://doi.org/10.2514/6.1994-56.

[61] Z. Biao, Q. Zhide, and G. Chao. Transonic flutter analysis of an airfoil with approximate boundary method. In *26th international congress of the aeronautical sciences*, 2008. http://www.icas.org/ICAS_ARCHIVE/ICAS2008/PAPERS/230.PDF, Accessed October 31, 2017.

[62] J.P. Thomas, K.C. Hall, and E.H. Dowell. Reduced-order aeroelastic modeling using proper-orthogonal decompositions. *Presented at CEAS/AI-*

AA/ICASE/NASA Langley International Forum on Aeroelasticity and Structural Dynamics, 1999. http://people.duke.edu/~jthomas/papers/papers/podairfoil.pdf, Accessed October 31, 2017.

[63] R. Sanchez, R. Palacios, T.D. Economon, H.L. Kline, J.J. Alonso, and F. Palacios. Towards a fluid-structure interaction solver for problems with large deformations within the open-source SU2 suite. In *AIAA 2016-0205. 57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, January, 4-8, 2016. https://doi.org/10.2514/6.2016-0205.

[64] C. Kassiotis, A. Ibrahimbegovic, R. Niekamp, and H. Matthies. Nonlinear fluid-structure interaction problem. Part I : implicit partitioned algorithm, nonlinear stability proof and validation examples. *Computational Mechanics*, 47(3):305–323, 2011. https://doi.org/10.1007/s00466-010-0545-6.

[65] M. Olivier, G. Dumas, and J. Morissette. A fluid-structure interaction solver for nano-air-vehicle flapping wings. In *AIAA Paper 2009-3676. 19th AIAA Computational Fluid Dynamics Conference*, pages 1–15, San Antonio, USA, June 2009. https://doi.org/10.2514/6.2009-3676.

[66] E.C. Yates. AGARD standard aeroelastic configuration for dynamic response I - Wing 445.6. *AGARD Report 765*, 1988. http://www.dtic.mil/dtic/tr/fulltext/u2/a199433.pdf, Accessed October 31, 2017.

[67] G.S.L. Goura. *Time marching analysis of flutter using computational fluid dynamics*. PhD thesis, University of Glasgow, 2001.

[68] R.J. Beaubien, F. Nitzsche, and D. Feszty. Time and frequency domain solutions for the AGARD 445 wing. In *International Forum on Aeroelasticity and Structural Dynamics (IFASD)*, Munich, Germany, 2005. https://www.researchgate.net/publication/228737999_Time_and_frequency_domain_flutter_solutions_for_the_AGARD_4456_wing, Accessed October 31, 2017.

[69] B. Zhang, W. Ding, J. Shengcheng, and J. Zhang. Transonic flutter analysis of an AGARD 445.6 wing in the frequency domain using the Euler method. *Engineering applications of computational fluid mechanics*, 10(1):244–255, 2016. http://dx.doi.org/10.1080/19942060.2016.1152200.

[70] E.M. Lee-Rausch and J.T. Batina. Calculation of AGARD wing 445.6 flutter using Navier-Stokes aerodynamics. In *AIAA paper 93-3476. 11th Applied Aerodynamics Conference*, Monterey, CA, USA, 1993. https://doi.org/10.2514/6.1993-3476.

[71] J. Xiao and C. Gu. Wing flutter simulations using an aeroelastic solver based on the predictor-corrector scheme. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 224(11): 1193–1210, 2010. https://doi.org/10.1243/09544100JAERO756.

[72] M. Luhar and H.M. Nepf. Flow-induced reconfiguration of buoyant and flexible aquatic vegetation. *Limnology and Oceanography*, 56(6):2003–2017, 2011. https://doi.org/10.4319/lo.2011.56.6.2003.

[73] K. Taira and T. Colonius. Three-dimensional flows around low-aspect-ratio flat-plate wings at low Reynolds numbers. *Journal of Fluid Mechanics*, 623: 187–207, 2009. https://doi.org/10.1017/S0022112008005314.