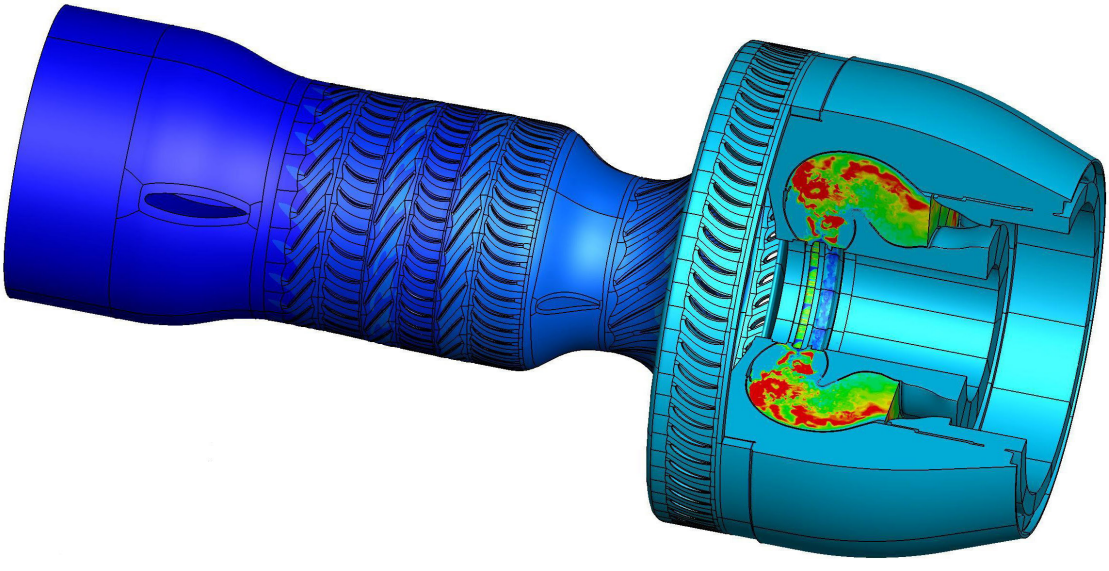
Comptes Rendus
de l'Académie des sciences

1873-7234 (electronic)

Mécanique



Thematic issue / *Numéro thématique*
Multiphysics simulation environments / *Les environnements de simulation multiphysiques*

Coordinated by / *Coordonné par*
Nicolas Berthier, Denis Gueyffier, Éric Quémerais

cnrs

INVENIT ET PERFICIT 1666

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

CENTRE MERSENNE

Comptes Rendus
Mécanique

Objective of the journal

Comptes Rendus Mécanique is an international peer-reviewed electronic journal, covering all areas of mechanics and engineering science.
It publishes original research articles, review articles, historical perspectives, pedagogical texts or conference proceedings, of unlimited length, in English or in French and in as flexible a format as necessary (figures, associated data, etc.).
Comptes Rendus Mécanique has been published since 2020 with the centre Mersenne pour l'édition scientifique ouverte (Mersenne Center for open scientific publishing), according to a virtuous Diamond Open Access policy, free for authors (no author processing charges nor publishing fees) as well as for readers (immediate and permanent open access).

About the journal

All journal's information, including the text of published articles, which is fully open access, is available from the journal website at https://comptes-rendus.academie-sciences.fr/mecanique/.

Author enquiries

For enquiries relating to the submission of articles, please visit this journal's homepage at https://comptes-rendus.academie-sciences.fr/mecanique/.

Contact

Académie des sciences
23, quai de Conti, 75006 Paris, France
Tel: (+33) (0)1 44 41 43 72
cr-mecanique@academie-sciences.fr

To cite this issue:
Nicolas Berthier, Denis Guyeffier, Éric Quémerais (ed). Multiphysics simulation environments. Comptes Rendus Mécanique, 2026. https://doi.org/10.5802/crmeca.sp.4.

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus. Mécanique
2025, Vol. 353, p. 1157-1159
https://doi.org/10.5802/crmeca.326

Foreword / *Avant-propos*

Introduction: from pioneering works to nowadays cutting-edge multiphysics simulations

Introduction : des travaux pionniers aux simulations multiphysiques de pointe actuelles

Nicolas Bertier [a], Denis Gueyffier [b] and Eric Quémerais [c]

[a] DMPE, ONERA, Université de Toulouse, 31000 Toulouse, France

[b] DSG, ONERA, Université Paris-Saclay, 91120 Palaiseau, France

[c] DMPE, ONERA, Université Paris-Saclay, 92320 Châtillon, France

E-mails: nicolas.bertier@onera.fr, denis.gueyffier@onera.fr, eric.quemerais@onera.fr

This issue explores the shift in the field of numerical simulation towards developing increasingly complex software platforms and frameworks. Among the reasons which led to this trend is the need to simulate complex industrial systems (e.g. airplanes, rockets, power plant...), systems in extreme conditions involving a large number of physics (e.g. atmospheric reentry, plasma assisted ignition, nuclear fusion...) or the need to create simulation software leveraging the potential and power of next generation supercomputers. We have gathered in this issue scientific articles showing the rich interplay between the sciences and techniques of numerical simulation, high-performance computing and software engineering necessary to build simulation frameworks for solving the multiphysics problems which are still intractable today. Advanced applications achieved with these recent tools are also presented in fields such as aerodynamics, energy, and in domains requiring the resolution of coupled multiphysical problems.

Before introducing the issue's papers, we describe, starting from the early days of numerical simulation, how the scientific community has made use of increasing computer power along with advanced simulation software to solve more and more complex problems. New algorithms and software architecture developed in the meantime has fueled the creation of such cutting-edge simulation software.

Although many mathematical foundations already existed, modern numerical simulation is often said to begin with the pioneering work of John Von Neumann at Los Alamos during the Manhattan Project (from 1942 to 1946). The objective was to simulate nuclear explosions, leading to numerous foundational works in numerical fluid mechanics, particularly on compressible fluid dynamics. The wartime effort also led to the invention of early computers in several countries (Z3 in Germany, Colossus in England, Harvard Mark I in USA...) and under the impulsion of Von Neumann, the US Army's ENIAC became the first computer with memory

storage capacity. At that time, models were extremely simplified, often associated with one-dimensional approaches, and programs were written in machine languages (binary, octal, or decimal) using punched cards as physical support.

The 1950s saw the establishment of several foundational elements of numerical simulation, including the development of finite element and finite volume methods, the emergence of computers with more advanced memory storage (IBM 701 in 1952 with a Von Neumann's architecture), and the advent of the Fortran language (Formula Translation, 1957). Several specialized calculation codes (in thermal, structural, and fluid mechanics, acoustics...) were developed in the 1960s–80s, preceding the rise of multiphysical numerical simulation as we know it today and the development of commercial software. In parallel, computing power increased considerably, roughly following Moore's Law ("The number of transistors in an integrated circuit doubles approximately every 18 to 24 months at constant cost"). The late 70s and early 80s saw the rise of the Cray family of supercomputers (Cray 1, Cray X-MP, Cray 2...) using vector processors and shared memory. The release of the first Silicon Graphics SGI station (SGI IRIS 1000) in 1984 enabled significant advancements in visualization and post-processing of simulation results. Researchers started tackling three-dimensional problems using these computers.

The 2000s marked the entry into the era of distributed memory clusters which enabled multiphysics calculations (notably using MPI protocol) on an increasingly large number of cores. This era led to better exploitation of processor chip architecture (memory levels and vectorized floating-point operation registers). But, during this period, the trend of increasing processor speeds was replaced with increasing the number of processor cores. With the development of GPU, today's supercomputers are hybrid clusters with multicore processors and GPU accelerators. All these scientific and technical advances have led to a complexification of numerical tool development, profoundly modifying the way numerical simulation tools are designed. While, until the 1990s, each numerical simulation software could still be developed by one person, teams with specialized profiles (experts of physical modeling, numerical method specialists, computer scientists, HPC gurus...) were gradually formed. As application complexity and tool development increased, it became increasingly difficult for a single team to possess the critical mass and expertise required to develop a tool with all the needed functionalities. This led to the development of more elaborate simulation platforms with modular design, where each software component can be developed by a different, specialized team. Each component can benefit from algorithms, programming techniques, and languages best adapted to both usage and the developing team's culture. The complexity is partially shifted to inter-component communication, raising coupling and data model problems. New architectures simplify code maintenance and addition of new functionalities. In 2025, we are at a true turning point in the discipline of numerical simulation. As several articles in this special issue attest, great maturity has been achieved in multiphysics platform development and in related coupling challenges.

In this issue, de Chaisemartin et al. present "ArcNum", an Arcane-based numerical framework, and its application to porous media flow simulation. In a different field, Antoine et al. focus on aeroelasticity with a strong emphasis on coupling challenges. Simon et al. discuss high-order adaptive multistep coupling algorithms, with an application to partial differential equations. In addition, a particularly important challenge for coupling is to provide efficient geometric tools on parallel hardware architectures. This aspect is highlighted by Andrieu et al., with a work on dynamic load-balanced point location algorithm for data mapping. Finally, considering the full complexity of most advanced applications while limiting CPU cost requires the use of advanced features such as dynamic mesh adaptation. Lugrin et al.'s work on a canonical dual mode scramjet isolator is a fine illustration of this aspect.

We can now seriously consider the advent of genuine "digital twins", using current tools. However, since 2012 and the deep learning revolution, artificial intelligence and its applications have

experienced exponential growth. Its incursion into numerical simulation is likely to revolutionize the discipline. We believe that machine learning will become pervasive in the field of numerical simulation, as can be attested by the rise of state of the art machine learning models like Aurora, GraphCast and GenCast in the field of weather prediction, challenging classical atmosphere simulation models. Substantial advances can be imagined in modeling (turbulence, combustion, liquid atomization...), in augmenting and replacing space and time discretization methods or in the way code is generated (using copilot-like assistants, on-the-fly documentation, automatic verification and optimization...).

Another consequence of the massive training of large language models and multimodal generative AI is the exploding demand for GPUs together with the multiplication of chips vendors. The rise of GPUs makes it essential for numerical simulation tools to adopt versatile architectures and to enable efficient use on both CPUs and GPUs. This has been performed elegantly using code generation among other techniques as shown in the the paper of Lienhardt et al., "SONICS: design and workflow of a new CFD software".

Two important trends in the field of numerical simulation are not addressed in this issue. The first one is cloud computing, as we believe it will greatly democratize access to HPC resources (e.g., small businesses performing high-fidelity simulations without supercomputer infrastructure) and improve user experience. The second one is the development of quantum computers. In the long term, they will likely become the backbone for solving numerical simulation problems, but this entails a completely different approach to numerical methods for solving PDEs than the ones which have persisted since the first CPUs, making it a significant challenge. These two trends will certainly be the subject of intense scientific and technical efforts in the community, and may become the subject of a future issue of *Comptes Rendus de l'Académie des Sciences*.

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus. Mécanique
2025, Vol. 353, p. 1159-1184
https://doi.org/10.5802/crmeca.333

Research article

High-order multistep coupling: convergence, stability and PDE application

Antoine E. Simon [*,a,b], **Laurent François** [a] **and Marc Massot** [c]

[a] ONERA, 6 Chemin de la Vauve aux Granges, 91123 Palaiseau, France

[b] CMAP, CNRS — École polytechnique, Institut Polytechnique de Paris, Route de Saclay, 91120 Palaiseau Cedex, France

E-mails: antoine.simon@polytechnique.edu, laurent.francois@onera.fr, marc.massot@polytechnique.edu

Abstract. Designing coupling schemes for specialized advanced mono-physics solvers in order to conduct accurate and efficient multiphysics simulations is a key issue that has recently received a lot of attention. A novel high-order adaptive multistep coupling strategy has shown potential to improve the efficiency and accuracy of such simulations, but requires further analysis. The purpose of the present contribution is to conduct the numerical analysis of convergence of the explicit and implicit variants of the method and to provide a first analysis of its absolute stability. A simplified coupled problem is constructed to assess the stability of the method along the lines of the Dahlquist's test equation for ODEs. We propose a connection with the stability analysis of other methods such as splitting and ImEx schemes. A stability analysis on a representative conjugate heat transfer case is also presented. This work constitutes a first building block to an a priori analysis of the stability of coupled PDEs.

Keywords. Multiphysics simulation, high-order, multistep coupling, stability, conjugate heat transfer.

Funding. The main support of ONERA and the Exa-MA project (Methods and Algorithms for Exascale, part of the PEPR NumPEx, Grant/Award Number: ANR-22-EXNU-0002 — https://numpex.org/) is gratefully acknowledged. The support of Fondation de l'École polytechnique through Initiative HPC@Maths is also gratefully acknowledged.

Note. Article submitted by invitation.

Manuscript received 20 January 2025, revised 17 September 2025, accepted 20 October 2025.

1. Introduction

The study of complex systems requires simulating the interactions between various physical phenomena involving multiple scales in time and space. The simulation of such multiphysics systems seldom relies on implementing a new solver for the coupled problem since each physics already possesses optimized and specialized solvers with an important and long-term legacy. As a consequence, multiphysics simulations are often built by coupling multiple existing solvers, which resolve separately each submodel and regularly exchange some quantities (volume or surface coupling terms) that we will refer to as *coupling variables*. A wide range of applications resort to this approach, e.g. fluid-structure interactions [1], fluid-plasma interactions [2], conjugate heat transfer [3,4], fluid-fluid coupling for acoustics [5].

So far, the key effort has been put on developing high-performance-computing (HPC) coupling libraries specialised in data transfer and spatial interpolation from one mesh to the other,

* Corresponding author

e.g. CWIPI [6,7], and has been applied in an industrial context [8]. The time integration is however often basic, with the codes being coupled at a fixed rate (constant coupling time step), and with the exchanged quantities being frozen over each coupling step [9]. This induces a first-order overall global error in the coupled dynamics. Besides, in most cases the coupling scheme is explicit since an implicit coupling is more difficult to implement and generally requires an iterative procedure. Consequently, for each simulation case, a fair amount of work is needed to determine a coupling time step that leads to a stable and precise coupled computation, which may be even more difficult if the dynamics and time scales strongly evolve with time. Thus, there is a need for a more efficient temporal coupling scheme, ensuring high-order accuracy in time, automatic adaptation of the coupling time step, and the ability to perform an implicit coupling. Solutions mimicking operator splitting [1], although easy to implement and fairly stable despite their explicit nature, cannot go beyond second-order accuracy. Implicit-Explicit (ImEx) schemes [10] have been originally designed for the case where multiple operators, e.g. convection and diffusion, act on a single physical system and spatial domain. They can however be advantageously applied as coupling schemes [3], solving implicitly the uncoupled dynamics of each coupled model, and handling the coupling terms explicitly. Still, ImEx schemes require the use of the same implicit time integration scheme for all solvers and prevent subcycling. Moreover, the coupling remains explicit and thus prone to instability for strongly-coupled systems.

Multirate time schemes have a long standing history [11] since their introduction in [12,13]. They rely on a separation of time scales between different terms and introduce a fixed time step ratio that prescribes the amount of subcycling performed for one term compared to the others. The approach has been used both for coupling [14] and for treating several coupled operators within the same domain simulation [15,16] in mono-physics solvers. This approach is however more suited to mono-physics solvers as they require some important modifications to the time integration procedure. Moreover, the amount of subcycling cannot be changed, thus making it difficult to cope with configurations where two coupled models may have similar or evolving time scales, and/or have a varying coupling strength as time evolves.

An interesting alternative approach has been developed in the coupling library PRECICE [17], in which the coupling variables are interpolated within a coupling time step, based on their values at the substeps of the different solvers. This enables a high-order implicit coupling in time with arbitrary subcycling. However, an explicit variant relying on extrapolation from the last coupling step is not available, and the coupling time step cannot be dynamically adapted.

A new generic coupling strategy has been recently introduced to remedy these problems [18–20], where the coupling variables are approximated by high-order polynomials of time during a coupling time step, enabling each solver to have accurate exchange terms during their internal substepping. These polynomials are built by an interpolation procedure involving the values of these variables at the previous coupling times. The polynomial representation is similar to that used in some co-simulation algorithms [21,22], which however are more focused on mechanical systems and assume a different structure for the coupling variables. The multistep coupling approach allows each solver of each subsystem to use any time integration scheme, potentially involving subcycling. The polynomial representation provides high orders of convergence and allows for the derivation of error estimates to dynamically adapt the coupling time step. Both explicit and implicit formulations of the coupling are available. Owing to its similarity with multistep schemes for ordinary differential equations (ODEs), this scheme is named *multistep coupling*, even if our method does not fall into the class of classical linear multistep methods.

A natural question, while examining the literature, is to investigate if the multistep coupling can be thought of as a domain decomposition method in space-time, where the space decomposition is prescribed by the physical domains associated with each model. The time decomposition could consist of (potentially adaptive) discretisation of the overall time interval into multiple

non-overlapping coupling time windows, solved in a sequential manner, each time with a high-order approximation of the coupling conditions (transmission conditions). However, the features of our scheme (non-monolithic method with black-box solvers for each subdomain involving different time integrators and substeps, different models and non-overlapping domains) imply that it does not fall into any of the traditional frameworks of domain decomposition, which are focused on the efficient parallelisation of monolithic-like implicit schemes or steady-state problems, such as domain overlap, optimised transmission conditions and coarse spaces (see [23,24] and references therein). Our method thus requires a dedicated numerical analysis investigation.

This multistep approach is promising, all the more if integrated within existing coupling libraries such as CWIPI. However, while for ODE integration schemes, be it one-step or multistep methods, the notion of absolute stability is well-defined through Dahlquist's test equation [25] linked to an eigenvalue analysis of the Jacobian matrix, no such framework exists for coupling schemes. In general, a much more involved study is conducted, where stability properties are determined numerically through a large parametric study on simple test cases [22].

The purpose of this contribution is to conduct the numerical analysis of the multistep coupling scheme in terms of order and convergence, and to propose a framework for the stability analysis of coupling schemes, in the same spirit as Dahlquist's test equation for ODE solvers. It provides a first building block for a connection with coupled partial differential equations (PDEs), as encountered in practical applications, and paves the way for a complete a priori analysis of coupling stability.

The contribution is organized as follows. In a second section after the introduction, the high-order multistep coupling is presented and the consistency/order and convergence analysis conducted. In a third section, we present a simple coupled problem in order to numerically illustrate the convergence properties. It turns out that this simple problem is an interesting generalisation to coupled systems of the well-known Dahlquist's test equation. Its use for the construction of stability diagrams is discussed in a fourth section, were a comprehensive stability analysis is conducted. We also conduct a comparison between the multistep coupling scheme and some splitting [26] and ImEx Additive Runge–Kutta (ARK) schemes [10,27] from the literature. The last section presents a numerical stability analysis of a conjugate heat transfer problem, and discussions on the link between the test equation and the conjugate heat model are presented in a conclusion.

2. The multistep coupling scheme

2.1. *Explicit and implicit formulations of the multistep coupling scheme*

The multistep coupling scheme is a strategy to integrate a set of coupled ODEs, in particular those arising from the spatial semi-discretisation of coupled PDEs. For such systems, the global state vector $y\colon [0,T] \to \mathbb{R}^m$ with $T \in \mathbb{R}_+^*$ and $m \in \mathbb{N}^*$ can be formally decomposed into components as $y = (y_1^t \cdots y_M^t)^t$ ($M \in \mathbb{N}^*$). The component $y_i\colon [0,T] \to \mathbb{R}^{m_i}$ is the state vector of the i-th subsystem (associated with one particular solver in the multistep coupling scheme) with $m_i \in \mathbb{N}^*$. In the framework of coupled equations arising from a multiphysics problem, the dynamics of each subsystem $i = 1,\dots,M$ depends both on the state vector y_i and on the *coupling variables* $u_i\colon [0,T] \to \mathbb{R}^{l_i}$ ($l_i \in \mathbb{N}^*$) which may depend on all subsystems $j = 1,\dots,M$. We assume the form $u_i = h_i(y_1,\dots,y_M)$, with h_i a Lipschitz function. In general, the coupling involves only a lower-dimensional interface between spatial domains so that in most cases $l_i \ll m_i$, e.g. coupling a

3D fluid flow to a 3D wing model by pressure and displacement terms defined on the 2D wing surface. The whole coupled system's dynamics is described by:

$$\begin{cases} \mathrm{d}_t y_i(t) = \mathscr{F}_i\big(y_i(t), u_i(t), t\big) \\ u_i = h_i(y_1, \dots, y_M) \\ y_i \in \mathscr{C}^1\big([0,T], \mathbb{R}^{m_i}\big) \\ y_i(0) = y_{i,0} \in \mathbb{R}^{m_i} \end{cases} \qquad \text{for } i = 1, \dots, M \tag{1}$$

where, to ensure well-posedness of this equation, $\mathscr{F}_i : \mathbb{R}^m \times [0,T] \to \mathbb{R}^m$ are continuous functions, Lipschitz in terms of y_i and u_i, and h_i are Lipschitz functions. For each i, the problem described in equation (1) is called a mono-physics problem, or subsystem.

The multistep coupling scheme consists in building, for the n-th coupling step $[t_n, t_{n+1}]$, polynomial predictions $\widehat{u}_i^n : [t_n, t_{n+1}] \to \mathbb{R}^{l_i}$ of $u_i(t)$, $i = 1, \dots, M$. These predictors are obtained, in the explicit case, by an extrapolation of the past states u_{n-j}, $j \in \{0, \dots, k\}$, $k \in \mathbb{N}$. The prediction operators are denoted Ψ_i. In the explicit case, they are defined by:

$$\Psi_i : \begin{cases} (\mathbb{R}^{l_i})^{k+1} \times [0,T]^{k+1} \longrightarrow \mathscr{C}^\infty\big([t_n, t_{n+1}], \mathbb{R}^{l_i}\big) \\ (u_i^{n-k}, \dots, u_i^n, t_{n-k}, \dots, t_n) \longmapsto \widehat{u}_i^n \end{cases} \tag{2}$$

where

$$\widehat{u}_i^n(t) = \Psi_i(u_i^{n-k}, \dots, u_i^n, t_{n-k}, \dots, t_n)(t) = \sum_{j=0}^{k} u_i^{n-j} \prod_{\substack{l=0 \\ l \neq j}}^{k} \frac{t - t_{n-l}}{t_{n-j} - t_{n-l}}. \tag{3}$$

These predictors allow to integrate all subsystems independently over a coupling time step of size Δt_n:

$$\mathrm{d}_t y_i(t) = \mathscr{F}_i\big(y_i(t), \widehat{u}_i^n(t), t\big) \tag{4}$$

for all $i = 1, \dots, M$, using M mono-physics solvers. These solvers can use any reasonable time integration technique, potentially involving subcycling.

As mentioned in the introduction, the simplest and ubiquitous coupling scheme consists in setting $\widehat{u}_i^n(t) = u_i^n$. This corresponds to a constant extrapolation ($k = 0$) in equation (3). This strategy results in a first-order global error. We will show in Theorem 9 that the multistep coupling technique has a global error of order $k+1$, with k the polynomial degree of the predictions. Thus, it provides a relatively simple way to improve the time accuracy of a coupled simulation.

The definition of $\widehat{u}_i$ in equation (3) leads to an explicit scheme, since data is produced by extrapolating from the past. An implicit formulation can be obtained by using the future (and yet unknown) value u_i^{n+1} as an additional node to define $\widehat{u}_i^n$, so that Ψ_i becomes an interpolant of u_i:

$$\widehat{u}_i^n(t) = \Psi_i(u_i^{n-k+1}, \dots, u_i^{n+1}, t_{n-k+1}, \dots, t_{n+1})(t).$$

Applying this to all subsystems, a nonlinear system on u_i^{n+1}, $i = 1, \dots, M$, is obtained, which can be solved by Picard iterations or Newton-like schemes [28]. Some model-specific techniques stemming from the domain decomposition community [23] may also be applied to precondition the fixed-point operator, for instance by adopting optimized Robin-type coupling conditions if applicable. The solution method for the fixed-point problem in the implicit case is however beyond the scope of the present paper.

In the following, it will be useful to recast the multistep coupling scheme as a one-step method by writing it in vector form. For $n = 0, \dots, N-1$, the overall state vector at time t_n is denoted $y^n = \big((y_1^n)^t, \dots, (y_M^n)^t\big)^t$ and the concatenation of the last $k+1$ values of y at times $t_n, \dots, t_{n-k}$ is:

$$Y_n = \big((y^n)^t, \dots, (y^{n-k})^t\big)^t \in E \tag{5}$$

where $E = (\mathbb{R}^m)^{k+1}$. We denote as Φ_i, $i = 1, \dots, M$, the mono-physics integrators associated with each mono-physics problem, which we assume to be zero-stable as will be defined in

Section 2.2.1. The integrator Φ_i corresponds to the application of the i-th solver for the time integration of equation (4) for subsystem i over a coupling time step:

$$\Phi_i : \begin{cases} [0,T] \times \mathbb{R}^{m_i} \times \mathscr{C}^0\big([t_n, t_{n+1}], \mathbb{R}^{l_i}\big) \times]0,T] \longrightarrow \mathbb{R}^{m_i} \\ \qquad\qquad (t, y_i^n, \widehat{u}_i^n, \Delta t_n) \longmapsto \Phi_i(t, y_i^n, \widehat{u}_i^n, \Delta t_n) \end{cases} \tag{6}$$

such that, for a time step Δt_n, the multistep coupling procedure is:

$$y_i^{n+1} = y_i^n + \Delta t_n \Phi_i(t_n, y_i^n, \widehat{u}_i^n, \Delta t_n). \tag{7}$$

This last equation allows to rewrite the whole scheme as follows:

$$\begin{cases} Y_{n+1} = Y_n + \Delta t_n \Phi(t_{n-k}, \ldots, t_n, Y_n, \Delta t_n) & \text{(explicit)} \\ Y_{n+1} = Y_n + \Delta t_n \Phi(t_{n-k+1}, \ldots, t_{n+1}, Y_{n+1}, \Delta t_n) & \text{(implicit)} \end{cases} \tag{8}$$

where Φ denotes the concatenation of all Φ_i operators composed with the Ψ_i operators. Note that the initial value y_n is contained within Y_{n+1}, thus Φ need not depend on Y_n for the implicit case. Assuming a solution to the implicit formulation exists (see Lemma 4), the implicit scheme can be written in the usual one-step form:

$$Y_{n+1} = Y_n + \Delta t_n \widetilde{\Phi}(t_{n-k}, \ldots, t_n, Y_n, \Delta t_n) \tag{9}$$

where $\widetilde{\Phi}$ involves the corresponding solution operator.

Note that the multistep coupling scheme is not self-starting for $k > 0$ in the explicit case or $k > 1$ in the implicit case. Just as for multistep ODE integrators, multiple solutions can be considered. A first one is to perform the first k coupling steps with another high-order coupling scheme. Another solution is to simply start at $k = 0$ with a very small time step, and increase k (and potentially the time step) at each subsequent step, until the targeted value of k is reached. A final solution for the implicit scheme is to perform the first k steps in a strongly coupled manner, i.e. solving a more complex interpolation problem involving the first k values of the coupling variables, used to define a single high-order interpolant for these first steps.

2.2. *Convergence of the multistep coupling scheme*

We now demonstrate that the multistep coupling scheme, either in explicit or implicit form, is zero-stable and consistent as long as each mono-physics integrator Φ_i, $i = 1, \ldots, M$, satisfies the usual Lipschitz properties leading to their zero-stability. Convergence of the scheme can then be obtained. The result is valid for exact or approximate subsystem integration. In this section, we assume that the coupling time step is constant, set to Δt. The overall simulation time is $T = N\Delta t$, where $N \in \mathbb{N}^*$. Using a constant coupling time step allows to simplify the dependency of operators defined above, Ψ_i then only depends on $(u_i^{n-k}, \ldots, u_i^n, t_n, \Delta t)$, and identically, Φ only depends on $(t_n, Y_n, \Delta t)$ and $\widetilde{\Phi}$ on $(t_n, Y_{n+1}, \Delta t)$.

2.2.1. *Zero-stability*

A scheme is said to be *zero-stable* if the amplification across multiple steps of perturbations is bounded according to the following definition.

Definition 1 (Zero-stability). *Let $\|\cdot\|$ be any norm on E, and φ be an integration scheme expressed as $Y_{n+1} = Y_n + \Delta t \varphi(t_n, Y_n, \Delta t)$. This scheme is said to be* zero-stable *if there exists a constant $K > 0$ independent of Δt such that $\forall\ Y_0, Z_0 \in E$ and $\forall\ (\varepsilon_n)_{n=0,\ldots,N-1} \in E^N$, the sequences:*

$$\begin{cases} Y_{n+1} = Y_n + \Delta t \varphi(t_n, Y_n, \Delta t) \\ Z_{n+1} = Z_n + \Delta t \varphi(t_n, Z_n, \Delta t) + \varepsilon_n \end{cases}$$

satisfy the following inequality:

$$\max_{0\leqslant n\leqslant N}\|Y_n - Z_n\| \leqslant K\left[\|Y_0 - Z_0\| + \sum_{n=0}^{N-1}\|\varepsilon_n\|\right]. \tag{10}$$

Note that this definition is classical for one-step schemes [29, Section 3.2] and may sometimes be referred to simply as *stability* [30, equation (16.1.7)]. When applied on the one-step reformulation of a linear multistep ODE scheme, this definition is equivalent to the traditional root condition from the linear multistep literature [30, Section (17.3)]. Since our scheme does not belong to the category of linear multistep methods, but is a generalisation thereof, the more general Definition 1 should be used. This definition can also be used more directly to study the convergence of the method (see Theorem 9.).

Let us stress that zero-stability is a prerequisite for an integrator to be convergent. As already mentioned, all subsystem integrators Φ_i are assumed to be Lipschitz operators and thus correspond to zero-stable time integrators.

To establish the zero-stability of the multistep coupling scheme, we first need to show that the multistep coupling integrator Φ defined by equation (8) constitutes a Lipschitz operator.

Lemma 2 (Φ is Lipschitz). *Provided that the integrators Φ_i are Lipschitz operators, the multistep coupling operator Φ from equation* (8) *is continuous and Lipschitz in Y.*

Proof. We first show that the extrapolation/interpolation operators Ψ_i are all Lipschitz in terms of their $k+1$ first variables. Let us consider $u_i^{n-k},\dots,u_i^n,\widetilde{u}_i^{n-k},\dots,\widetilde{u}_i^n \in \mathbb{R}^{l_i}$ and denote $u_i = \left((u_i^{n-k})^t,\dots,(u_i^n)^t\right)^t$ and $\widetilde{u}_i = \left((\widetilde{u}_i^{n-k})^t,\dots,(\widetilde{u}_i^n)^t\right)^t$. Consider the infinity norm $\|\cdot\|_\infty$ on the space $\mathscr{C}^0\left([t_n,t_{n+1}],\mathbb{R}^{l_i}\right)$, the infinity norm $\|\cdot\|_{\infty,\mathscr{I}}$ on the space $\mathscr{C}^0(\mathscr{I},\mathbb{R})$ where $\mathscr{I}$ is a closed interval of $\mathbb{R}$, and the infinity norm $|\cdot|_\infty$ on $(\mathbb{R}^{l_i})^{k+1}$. Using the dimensionless time $\tau = \frac{t-t_n}{\Delta t}$, we denote $\mathscr{L}_j, j = 0,\dots,k$ the basis of Lagrange interpolation polynomials at points $\tau_0 = 0$, $\tau_1 = -1$, …, $\tau_k = -k$. The explicit predictors are expressed as follows:

$$\widehat{u}_i(t) = \Psi_i(u_i^n,\dots,u_i^{n-k},t_n,\Delta t)(t) = \sum_{j=0}^{k} u_i^{n-j}\mathscr{L}_j(\tau)$$

with a similar expression for $\widehat{\widetilde{u}}_i(t)$. The following bound can be obtained directly:

$$\|\widehat{u}_i - \widehat{\widetilde{u}}_i\|_\infty \leqslant |u_i - \widetilde{u}_i|_\infty \cdot \sum_{j=0}^{k}\|\mathscr{L}_j\|_{\infty,[0,1]}.$$

Hence, all the Ψ_i are Lipschitz in terms of their $k+1$ first variables. Besides, the Lipschitz constant $\sum_{j=0}^{k}\|\mathscr{L}_j\|_{\infty,[0,1]}$ does not depend on Δt.

A similar result can be obtained for the case of interpolation, using $\widetilde{\mathscr{L}}_j$, $j = 0,\dots,k$ the basis of Lagrange interpolation polynomials at points $\tau_0 = 1, \tau_1 = 0, \dots, \tau_k = -k+1$.

Since all h_i and all integrators Φ_i are Lipschitz, then, Φ is also Lipschitz in terms of Y_n by composition of Lipschitz functions. □

The zero-stability of the explicit multistep coupling scheme can now be stated in the following proposition.

Proposition 3 (Zero-stability of the explicit multistep coupling scheme). *The explicit multistep coupling scheme given by equation* (8) *is zero-stable.*

Proof. As already stated, the subsystem integrators Φ_i are Lipschitz with a Lipschitz constant independent of Δt for all $0 < \Delta t \leqslant T$. Lemma 2 ensures that there exists a Lipschitz constant

$L_\Phi > 0$ of the overall integrator Φ independent of Δt. Consider $Y_0, Z_0 \in E$, $(\varepsilon_n)_{n=0,\dots,N-1} \in E^N$ and the sequences:

$$\begin{cases} Y_{n+1} = Y_n + \Delta t \Phi(t_n, Y_n, \Delta t) \\ Z_{n+1} = Z_n + \Delta t \Phi(t_n, Z_n, \Delta t) + \varepsilon_n. \end{cases}$$

Then, for any $n \leqslant N-1$:

$$\begin{aligned} |Y_{n+1} - Z_{n+1}|_\infty &\leqslant \Big| Y_n - Z_n + \Delta t \big[\Phi(t_n, Y_n, \Delta t) - \Phi(t_n, Z_n, \Delta t)\big] - \varepsilon_n \Big|_\infty \\ &\leqslant (1 + L_\Phi \Delta t) |Y_n - Z_n|_\infty + |\varepsilon_n|_\infty \\ &\leqslant (1 + L_\Phi \Delta t)^N |Y_0 - Z_0|_\infty + \sum_{i=0}^{N-1} (1 + L_\Phi \Delta t)^N |\varepsilon_i|_\infty \\ &\leqslant \underbrace{e^{L_\Phi T}}_{K} |Y_0 - Z_0|_\infty + \underbrace{e^{L_\Phi T}}_{K} \sum_{i=0}^{N-1} |\varepsilon_i|_\infty. \end{aligned}$$

□

To prove the same property for the implicit scheme, we first need to show that the implicit system in equation (8) is well-posed.

Lemma 4 (Well-posedness of the implicit multistep coupling scheme). *There exists $\Delta t^* > 0$ such that, for all $0 < \Delta t \leq \Delta t^*$ and for all $Y_n \in E$, there exists a unique solution Y_{n+1} to the non-linear equation $Y_{n+1} = Y_n + \Delta t \Phi(t_n, Y_{n+1}, \Delta t)$.*

Proof. Let Y_n be an element of E and $t_n \in [0, T[$. Lemma 2 ensures that there exists $L_\Phi > 0$ such that for all $\Delta t > 0$ and for all $Y, Z \in E$: $\big\|\Phi(t_n, Y, \Delta t) - \Phi(t_n, Z, \Delta t)\big\| \leq L_\Phi \|Y - Z\|$.

Then, for all $\Delta t > 0$ and for all $Y, Z \in E$:

$$\Big\| \big(Y_n + \Delta t \Phi(t_n, Y, \Delta t)\big) - \big(Y_n + \Delta t \Phi(t_n, Z, \Delta t)\big) \Big\| \leq \Delta t L_\Phi \|Y - Z\|.$$

Let Δt^* be a positive real number such that $\Delta t^* L_\Phi < 1$, Δt^* does not depend on Y_n. For all $0 < \Delta t \leq \Delta t^*$, the function $Y \mapsto Y_n + \Delta t \Phi(t_n, Y, \Delta t)$ is a contraction, therefore, it admits exactly one fixed point that we can denote Y_{n+1}. □

This proof also ensures that the previously introduced implicit integration operator $\widetilde{\Phi}$ is well defined. Note that the value of Δt^* and the constant K are related to the Lipschitz constant L_Φ, which is directly affected by the evaluation of the Lipschitz constant of the one-step ODE integrators Φ_i. For Δt^* and K to be close to optimality, the proper framework is that of the one-sided Lipschitz condition [25, Section IV.14], which is much more favourable in particular for contracting dynamics, the category into which dissipative systems fall.

The zero-stability of the implicit multistep coupling scheme can now be established.

Proposition 5 (Zero-stability of the implicit multistep coupling schemes). *The implicit multistep coupling scheme given by equation* (8) *is zero-stable for time steps ensuring its well-posedness.*

Proof. We consider $Y_0, Z_0 \in E$, $(\varepsilon_n)_{n=0,\dots N-1} \in E^N$ and the sequences defined below:

$$\begin{cases} Y_{n+1} = Y_n + \Delta t \Phi(t_n, Y_{n+1}, \Delta t) \\ Z_{n+1} = Z_n + \Delta t \Phi(t_n, Z_{n+1}, \Delta t) + \varepsilon_n \end{cases}$$

which are well-defined for all $0 < \Delta t < \Delta t^*$ according to Lemma 4. First, for any $0 < \Delta t < \Delta t^*$:

$$\frac{1}{1 - L_\Phi \Delta t} = 1 + \frac{\Delta t L_\Phi}{1 - L_\Phi \Delta t} \leq 1 + \Delta t K^*$$

where $K^* \equiv \frac{L_\Phi}{1-L_\Phi \Delta t^*}$. Then, for any $n \leqslant N-1$:

$$\begin{aligned}
|Y_{n+1} - Z_{n+1}|_\infty &\leqslant \Big| Y_n - Z_n + \Delta t\big[\Phi(t_n, Y_{n+1}, \Delta t) - \Phi(t_n, Z_{n+1}, \Delta t)\big] - \varepsilon_n \Big|_\infty \\
&\leqslant \frac{1}{1-L_\Phi \Delta t} |Y_n - Z_n|_\infty + \frac{1}{1-L_\Phi \Delta t} |\varepsilon_n|_\infty \\
&\leqslant \left(\frac{1}{1-L_\Phi \Delta t}\right)^N |Y_0 - Z_0|_\infty + \sum_{i=0}^{N-1} \left(\frac{1}{1-L_\Phi \Delta t}\right)^N |\varepsilon_i|_\infty \\
&\leqslant (1+\Delta t K^*)^N |Y_0 - Z_0|_\infty + \sum_{i=0}^{N-1} (1+\Delta t K^*)^N |\varepsilon_i|_\infty \\
&\leqslant \underbrace{e^{K^* T}}_{K} |Y_0 - Z_0|_\infty + \underbrace{e^{K^* T}}_{K} \sum_{i=0}^{N-1} |\varepsilon_i|_\infty .
\end{aligned}$$

□

2.2.2. *Consistency and convergence*

The accuracy of the solution computed with the multistep coupling scheme can now be assessed by considering its local and global errors, defined below. In this section, the consistency of the scheme is proved, which, combined with the previous results, establishes its convergence.

Definition 6 (Global and local error — convergence). *Let us consider the integration scheme equation* (8)*, y a solution of equation* (1) *and $Y(t) = \big(y(t)^t, y(t-\Delta t)^t, \ldots, y(t-k\Delta t)^t\big)^t$. Let $\|\cdot\|$ be any norm on E. The* global error *of the scheme is:*

$$e_n = Y(t_n) - Y_n$$

and the scheme is convergent *if:*

$$\lim_{\Delta t \to 0} \max_{n=0,\ldots,N} \|e_n\| = 0.$$

We call local error *or* local truncation error *the quantity:*

$$\mathscr{E}_n = Y(t_{n+1}) - Y(t_n) - \Delta t \Phi\big(t_n, Y(t_n), \Delta t\big).$$

Definition 7 (Consistency). *With the same hypothesis as in Definition 6 the scheme is said to be* consistent *if the local truncation error verifies:*

$$|\mathscr{E}_n| \leqslant \Delta t \epsilon(\Delta t)$$

with ϵ a function such that $\epsilon(\Delta t) \to 0$ when $\Delta t \to 0$.

Besides, if $\epsilon(\Delta t) = \mathscr{O}(\Delta t^p)$, the scheme is said to be consistent at order p.

Proposition 8 (Consistency of the multistep coupling scheme). *Assuming that the subsystem integrators Φ_i are consistent at order $p_i \geq k+1$, the multistep coupling scheme given in equation* (8) *is consistent and the order of consistency is $k+1$ where k is the degree of the polynomial predictor.*

The consistency has been proven to be of order $k+1$ prediction polynomials of degree k, assuming exact integration of the subsystems [18]. In the present contribution, we propose a more general proof considering non-exact mono-physics integrators Φ_i instead.

Proof. All $\mathscr{F}_i(y_i, u_i, t)$ are Lipschitz both in terms of y_i and u_i. Let us denote the corresponding Lipschitz constants $L_{i,y}$ and $L_{i,u}$. Consider $\widetilde{\widetilde{y}}_i^n \in \mathbb{R}^{m_i}$ the solution at time t_n. One step of the multistep scheme yields for the i-th subsystem:

$$\widetilde{\widetilde{y}}_i^{n+1} = \widetilde{\widetilde{y}}_i^n + \Delta t \Phi_i(t_n, \widetilde{\widetilde{y}}_i^n, \widehat{u}_i, \Delta t).$$

An exact integration of equation (4) with the same initial condition gives a solution $\widetilde{y}_i$:

$$\widetilde{y}_i(t) = \widetilde{\widetilde{y}}_i^n + \int_{t_n}^{t} \mathscr{F}_i\big(\widetilde{y}_i(s), \widehat{u}_i(s), s\big)\,\mathrm{d}s$$

where $\widehat{u}_i(s)$ is the polynomial prediction of the coupling variables. Finally, an exact integration of equation (1) with the same initial condition as above gives a solution y_i of the original problem such that:

$$y_i(t) = \widetilde{\widetilde{y}}_i^n + \int_{t_n}^{t} \mathscr{F}_i\big(y_i(s), u_i(s), s\big)\,\mathrm{d}s$$

where $u_i(s)$ is the exact evolution of the coupling variables.

Let us notice that $y_i(t_n) = \widetilde{y}_i(t_n) = \widetilde{\widetilde{y}}_i^n$.

The absolute local truncation error $|\mathscr{E}_n| = \big|\widetilde{\widetilde{y}}_i^{n+1} - y_i(t_{n+1})\big|$ then can be written:

$$\begin{aligned}\Big|y_i(t_{n+1}) - y_i(t_n) - \Delta t\Phi_i\big(t_n, y_i(t_n), \widehat{u}_i, \Delta t\big)\Big| &= \big|y_i(t_{n+1}) - \widetilde{\widetilde{y}}_i^{n+1}\big| \\ &= \big|y_i(t_{n+1}) - \widetilde{y}_i(t_{n+1}) + \widetilde{y}_i(t_{n+1}) - \widetilde{\widetilde{y}}_i^{n+1}\big| \\ &\le \underbrace{\big|y_i(t_{n+1}) - \widetilde{y}_i(t_{n+1})\big|}_{e_1} + \underbrace{\big|\widetilde{y}_i(t_{n+1}) - \widetilde{\widetilde{y}}_i^{n+1}\big|}_{e_2}\end{aligned}$$

where e_1 represents the error arising from the polynomial approximation of the coupling variables, and e_2 represents the error produced by the non-exact subsystem integration.

First, let us consider e_1:

$$\begin{aligned}e_1 &= \big|y_i(t_{n+1}) - \widetilde{y}_i(t_{n+1})\big| \\ &= \left|\widetilde{\widetilde{y}}_i^n + \int_{t_n}^{t_{n+1}} \mathscr{F}_i\big(y_i(s), u_i(s), s\big)\,\mathrm{d}s - \widetilde{\widetilde{y}}_i^n - \int_{t_n}^{t_{n+1}} \mathscr{F}_i\big(\widetilde{y}_i(s), \widehat{u}_i(s), s\big)\,\mathrm{d}s\right| \\ &\le \int_{t_n}^{t_{n+1}} \Big|\mathscr{F}_i\big(y_i(s), u_i(s), s\big) - \mathscr{F}_i\big(\widetilde{y}_i(s), \widehat{u}_i(s), s\big)\Big|\,\mathrm{d}s \\ &\le L_{i,u}\Delta t \max_{[t_n,t_{n+1}]} |u_i - \widehat{u}_i| + \int_{t_n}^{t_{n+1}} L_{i,y}\big|\widetilde{y}_i(s) - y_i(s)\big|\,\mathrm{d}s.\end{aligned}$$

Since $t \mapsto L_{i,u}\Delta t \max_{[t_n,t]}|u_i - \widehat{u}_i|$ is a non-decreasing function of t, Grönwall's inequality ensures that:

$$e_1 \le L_{i,u}\Delta t \max_{[t_n,t_{n+1}]} |u_i - \widehat{u}_i| \exp(L_{i,y}\Delta t).$$

Besides, the interpolation approximation error $\max_{[t_n,t]}|u_i - \widehat{u}_i|$ is of order $\mathscr{O}(\Delta t^{k+1})$ for an interpolation polynomial of degree k, hence:

$$e_1 = L_{i,u}\Delta t e^{L_{i,y}\Delta t}\mathscr{O}(\Delta t^{k+1}) = \mathscr{O}(\Delta t^{k+2}).$$

Then, consider e_2:

$$e_2 = \big|\widetilde{y}_i(t_{n+1}) - \widetilde{\widetilde{y}}_i^{n+1}\big| = \left|\widetilde{\widetilde{y}}_i^n + \int_{t_n}^{t_{n+1}} \mathscr{F}_i\big(\widetilde{y}_i(s), \widehat{u}_i(s), s\big)\,\mathrm{d}s - \widetilde{\widetilde{y}}_i^n - \Delta t\Phi_i(t_n, \widetilde{\widetilde{y}}_i^n, \widehat{u}_i, \Delta t)\right|.$$

Since the integrators Φ_i are consistent at order $p_i \ge k+1$, e_2 satisfies:

$$e_2 = \mathscr{O}(\Delta t^{p_i+1}) = \mathscr{O}(\Delta t^{k+2}).$$

Finally, $\Big|y_i(t_{n+1}) - y_i(t_n) - \Delta t\Phi_i\big(t_n, y_i(t_n), \widehat{u}_i, \Delta t\big)\Big| = \mathscr{O}(\Delta t^{k+2})$ and the scheme is consistent at order $k+1$. □

Thus, the order of consistency is the degree k of the prediction polynomials, and the subsystem integrators should be at least of order k to preserve the order of the coupling scheme. A formula to compute the error constant is derived in Appendix A. Much like multistep ODE integrators, increasing the order does not necessarily induce an increase in computational cost, but only an increase in storage for keeping past values of u_i. This is a strong advantage of the multistep coupling method.

We may now prove that the multistep coupling scheme satisfies Definition 6 and therefore is convergent.

Theorem 9 (Convergence of the multistep coupling scheme). *The multistep coupling scheme is convergent and converges at order $k+1$ for any $k \geq 0$.*

Proof. When plugging into Definition 1 of zero-stability $Y_0 = Z_0$ and $\varepsilon_n = \mathscr{E}_n$ (local truncation error from Definition 6), it can be seen that the sequence Z_n is the sequence $Z_n = Y(t_n)$ of the values of the exact solution of equation (1), while Y_n is the sequence of values given by the multistep coupling scheme. Then the results of Theorems 3, 5 and 8 give:

$$\left|Y_n - Y(t_n)\right| \leq K \underbrace{\left|Y_0 - Y(0)\right|}_{=0} + K \underbrace{\sum_{i=0}^{N-1} \underbrace{|\mathscr{E}_i|}_{=\mathscr{O}(\Delta t^{k+2})}}_{=\mathscr{O}(\Delta t^{k+1})} = \mathscr{O}(\Delta t^{k+1}).$$

Therefore, per Definition 6, we can conclude the proof. □

3. A simple model for the study of coupled equations

Now that we have demonstrated the previous essential theoretical properties of the scheme, we search for a simple coupled model to test the method, in particular to investigate its stability. In the literature, couplings are generally referred to as *weak* when stability is not a particular issue [3]. On the opposite, couplings inducing strong stability limits on explicit schemes are referred to as *strong* couplings [31,32]. However, no generic criterion has been proposed to predict how strong a coupling is.

In the context of co-simulation, the stability of coupling strategies is analyzed using chains of mass-spring systems [22]. Such test cases are already too complex, so that their stability can only be studied by a systematic numerical scan of the parameter space to locate stable set of parameter values and time step. This makes the analysis cumbersome and time-consuming, while also restricting the thorough understanding of coupling stability that can be obtained from it.

Having a complete theoretical picture of the coupling stability for generic coupled systems seems however out of reach. In the framework of ODEs, the Dahlquist's test equation $\mathrm{d}_t y = \lambda y$ is a well-established choice for the study of absolute stability and is built as follows. Any ODE $\mathrm{d}_t y = f(t, y)$ can be linearised locally along the solution path to obtain $\mathrm{d}_t y = (\partial_y f) y$. Diagonalisation of the Jacobian of f yields a system of scalar equations $\mathrm{d}_t x = \mu_i x$, where $\{\mu_i\}_i$ is the spectrum of the Jacobian. Stability is then ensured if the spectrum multiplied by the time step is contained within the stability domain of the chosen ODE integrator. The use of Dahlquist's test equation relies on a diagonalisation procedure of the linearised ODE, which is not relevant for coupled problems, since a global diagonalisation of the problem would destroy the partitioned nature of the approach. Hence a different test model should be sought.

In this section, we introduce a simple equation that aims at reproducing a coupled problem in its simplest form. Convergence results on this equation are presented to verify the convergence orders from Theorem 9. With certain parameter choice, stability seems to be an issue with the multistep explicit scheme, hence a deeper analysis of the scheme stability is proposed.

3.1. *The* 2×2 *test equation*

The simplest possible coupled problem is a coupling between two scalar equations as given in equation (11):

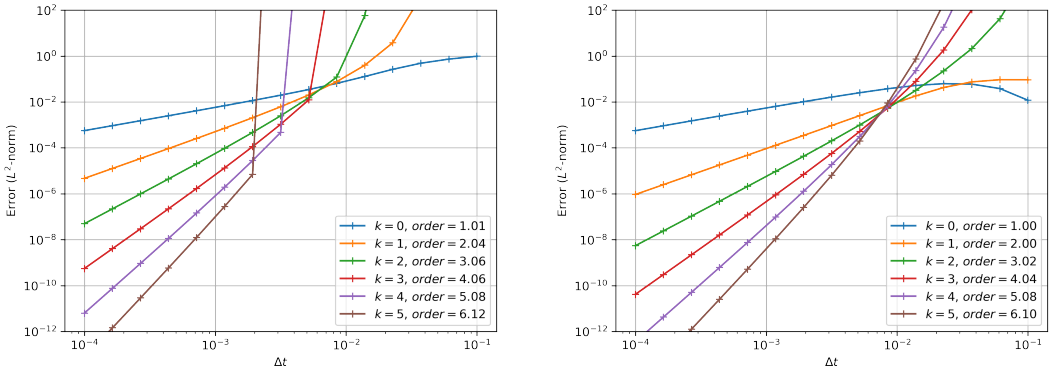
$$\begin{cases} \mathrm{d}_t y_1 = a_1 y_1 + a_{12} y_2 \\ \mathrm{d}_t y_2 = a_{21} y_1 + a_2 y_2 \end{cases} \iff \mathrm{d}_t \underbrace{\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}}_{\equiv y} = \underbrace{\begin{pmatrix} a_1 & a_{12} \\ a_{21} & a_2 \end{pmatrix}}_{\equiv \mathscr{A}} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \tag{11}$$

where $a_i \in \mathbb{C}$. We call this system the 2×2 *coupled test equation*. It corresponds to equation (1) with $M = 2$, $u_1 = h_1(y_1, y_2) = y_2$, $u_2 = h_2(y_1, y_2) = y_1$, $\mathscr{F}_1 \colon (y_1, u_1) \mapsto a_1 y_1 + a_{12} u_1$ and $\mathscr{F}_2 \colon (y_2, u_2) \mapsto a_{21} u_2 + a_2 y_2$. Variants of this model will be presented in Section 4.1.

When the multistep coupling scheme is applied to this model, the two scalar ODEs $\mathrm{d}_t y_1 = a_1 y_1 + a_{12} \widehat{u}_1$ and $\mathrm{d}_t y_2 = a_{21} \widehat{u}_2 + a_2 y_2$ are integrated in parallel, which is sometimes referred to as a Jacobi coupling [22]. At each coupling time step both predictors $\widehat{u}_1$ and $\widehat{u}_2$ are updated.

3.2. *Numerical convergence results*

We now use the previous model to check that the convergence rates established in Theorem 9 are obtained in practice. Both the implicit and explicit formulations are applied to the simple 2×2 coupled test equation, with parameters $a_{12} = 40$, $a_{21} = 40$, $a_2 = -80$ and $y(0) = (5, -2)^t$. Multiple fixed-time-step simulations are performed over the time interval $[0, 1]$ with different values of the time step. As discussed in Section 2.1, the scheme is not self-starting. Therefore, we initialise the prediction polynomials by using exact values of the coupling variables in negative time ($t_{-j} < 0$). Even though the system is dissipative, this does not produce any numerical artifacts for this simple model. The error is computed as the L^2-norm in time of the difference between these solutions and the analytical solution $\exp(At)y(0)$. The obtained convergence graphs are presented in Figure 1, where the numerically computed orders are indicated.



(a) Convergence of the explicit scheme

(b) Convergence of the implicit scheme

Figure 1. Convergence of the multistep scheme applied to the 2×2 coupled test equation with $a_1 = -80$, $a_{12} = 40$, $a_{21} = 40$, $a_2 = -80$ and $y(0) = (5, -2)^t$ integrated over the time interval $[0, 1]$.

All schemes reach their design order of convergence. The higher-order formulations provide an important decrease of the error, as expected. It may be noted that, for a given order $k > 1$, the implicit formulation possesses a lower error constant and is more precise. This is coherent with the error constants derived theoretically in Appendix A.

It should however be observed that the explicit multistep scheme, especially at higher orders, has an error which diverges when the time step is too large. It is actually related to a blow-up of the coupled solution. This instability phenomenon is investigated in the following section.

4. Stability analysis

It was observed in Section 3.2 that the multistep coupling scheme may suffer from instabilities. It would be interesting to be able to predict them from an a priori spectral analysis of the linearised

coupled system, as done for ODEs with Dahlquist's test equation. In this section, we first show that the 2×2 coupled test equation (11) is a relevant case for the study of coupled ODEs. We then recast the multistep scheme as a recurrence formula involving a matrix whose properties dictate the stability of the coupling, and we draw stability diagrams, which are compared with those from common splitting schemes and state-of-the-art ImEx ARK methods.

4.1. *On the characterisation of the stability domain*

One can notice that, assuming complex values for $\mathscr{A}$, the space of model parameters to analyze is of dimension 8. This makes it difficult to characterize intuitively and efficiently the complete stability domain of a coupling scheme. Therefore, two simplifying approaches are proposed to reduce the number of dimensions.

4.1.1. *Real asymmetrical case*

A first approach is to consider a general asymmetrical matrix $\mathscr{A}$ with real coefficients to reduce the number of dimensions to 4. Then, cross-sections of the parameter space can be made with a specific parametrization of $\mathscr{A}$, already used for the analysis of multirate schemes [33]. We introduce the following parameters:

$$\delta = \frac{a_2}{a_1} \qquad \text{and} \qquad \gamma = \frac{a_{12}a_{21}}{a_1 a_2}. \tag{12}$$

The parameter δ represents the ratio of the internal time scales of both scalar subsystems, while γ can be interpreted as a coupling strength, which measures how much the eigenvalues of the coupled problem depart from those of the scalar subsystems (i.e. from the diagonal of $\mathscr{A}$). These settings are relevant for several reasons. First, the stability of the continuous problem is guaranteed if and only if $a_1 + a_2 < 0$ and $\gamma < 1$. Besides, the stability of the multistep coupling scheme for a given Δt only depends on $(a_1 \Delta t, \delta, \gamma)$.[1] Thus, the use of γ reduces by 1 the number of dimensions, and two-dimensional cuts can be performed by specifying the value of δ. This approach is analysed in Section 4.4.

4.1.2. *Complex symmetrical case*

A second approach is to assume a symmetrical structure for $\mathscr{A}$ with equal diagonal terms. Considering complex coefficients, these assumptions on $\mathscr{A}$ reduce to 4 the number of dimensions for the parameter space. A cut can be performed (e.g. fixing real and imaginary parts of the diagonal term) to reduce it to a two-dimensional space in the spirit of ImEx schemes and thus allows to characterize it and use it more easily. This approach is developed in Section 4.5. The symmetrical coupled system can be linked to the generalised Dahlquist's test equation used in the study of ImEx schemes:

$$\mathrm{d}_t x = (a_1 + a_{12})x. \tag{13}$$

Indeed, assuming initial conditions $y_1(0) = y_2(0) = x(0)$, the solutions to both equations verify $y_1(t) = y_2(t) = x(t)$, $\forall\ t \geq 0$. The generalised Dahlquist's test equation actually corresponds to the solution of equation (11) along the eigenvector $(1,1)^t$ of $\mathscr{A}$ which is associated with the eigenvalue $a_1 + a_{12}$. Thus, stability domains obtained on both equations are similar, except near the stability limit of the continuous system, since the second eigenvector $(1,-1)^t$ of $\mathscr{A}$ (with eigenvalue $a_1 - a_{12}$) is not taken into account by the generalised Dahlquist's test equation.

Both real asymmetrical and complex symmetrical cases are relevant for our study of stability. On the one hand, the asymmetrical case allows to study the coupling of subsystems possessing

[1] This was confirmed for orders up to 15 using symbolic computation of the coefficients of the characteristic polynomial of the amplification matrix defined later in Section 4.2.

different internal time scales, and to see the effect of the off-diagonal coupling terms through a single parameter γ. On the other hand, the complex symmetrical case, limited to subsystems with identical time scales, allows to study the effect of imaginary parts in the coefficients. Combining both approaches will allow for a more complete picture of stability to be obtained.

4.2. *Recurrence matrix for the multistep coupling scheme*

In this section, we recast the multistep coupling method in one-step vectorial form. For a more generic formulation we temporarily consider the coupling of two ODE systems, of size m_1 and m_2. The case of the previous 2×2 coupled test equation can be easily recovered from the results presented in this section.

Let $m = m_1 + m_2$ and let $A_1 \in \mathbb{C}^{m_1 \times m_1}$, $A_2 \in \mathbb{C}^{m_2 \times m_2}$, $A_{12} \in \mathbb{C}^{m_1 \times m_2}$ and $A_{21} \in \mathbb{C}^{m_2 \times m_1}$ be complex-valued matrices. We consider the following linear coupled system:

$$\mathrm{d}_t y = Ay, \qquad A \equiv \begin{pmatrix} A_1 & A_{12} \\ A_{21} & A_2 \end{pmatrix} \tag{14}$$

where the first m_1 components of y constitute the state vector y_1 of the first subsystem, and the m_2 remaining ones constitute those of the state vector y_2 from the second subsystem. Let $\Delta t > 0$ be the time step used for our coupling scheme and $Z_i \equiv A_i \Delta t$. Then, for any degree $k \geq 0$, there exists a matrix $R(Z_1, Z_{12}, Z_{21}, Z_2, k)$ called the *recurrence matrix* or *amplification matrix* such that one step of the multistep coupling scheme can be expressed as:

$$Y_{n+1} = RY_n. \tag{15}$$

The coupling scheme is stable if the spectral radius $\rho(R)$ is lower than 1. In the case of the 2×2 coupled test equation, the same expressions of amplification matrices hold with $z_i = a_i \Delta t$.

The stability analysis will be conducted assuming an exact integration of the subsystems, so as to avoid making particular assumptions on the methods used for the subsystems integration. Although not presented here, numerical and theoretical stability analyses with non-exact integration have shown that using a non-subcycled low-order implicit scheme for the subsystem integration may increase the stability of the coupling scheme at the expense of accuracy.

Introducing $\tau = \frac{t - t_n}{\Delta t}$, the exact integration of equation (4) for the first subsystem yields:

$$y_1^{n+1} = e^{Z_1} y_1^n + \int_0^1 e^{Z_1(1-\tau)} Z_{12} \widehat{u}_1^n(t_n + \tau \Delta t)\, \mathrm{d}\tau. \tag{16}$$

For polynomial predictions of degree k, the amplification matrix of the explicit multistep scheme is:

$$R_{\mathrm{MS,exp}} = \begin{pmatrix} e^{Z_1} & S_1^0 & 0 & S_1^1 & \cdots & 0 & S_1^k \\ S_2^0 & e^{Z_2} & S_2^1 & 0 & \cdots & S_1^k & 0 \\ I_{m_1} & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & I_{m_2} & \ddots & & & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & I_{m_1} & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & I_{m_2} & 0 & 0 \end{pmatrix} \in \mathbf{C}^{m(k+1) \times m(k+1)}. \tag{17}$$

The basis of Lagrange interpolation polynomial at points $\tau_0 = -k$, $\tau_1 = -k+1$, …, $\tau_k = 0$ is denoted $\{\mathscr{L}_0, \ldots, \mathscr{L}_k\}$ so that the values of S_1^j and S_2^j are given by:

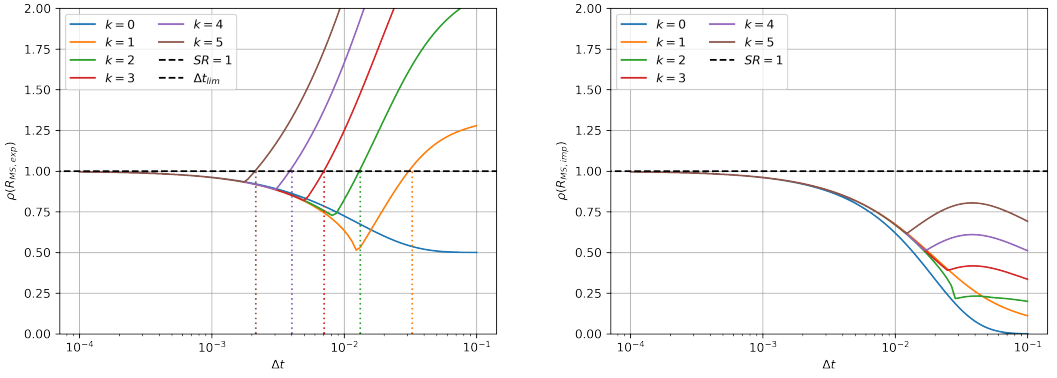
$$S_1^j = \int_0^1 e^{Z_1(1-\tau)} \mathscr{L}_j(\tau)\, \mathrm{d}\tau \cdot Z_{12} \qquad \text{and} \qquad S_2^j = \int_0^1 e^{Z_2(1-\tau)} \mathscr{L}_j(\tau)\, \mathrm{d}\tau \cdot Z_{21}. \tag{18}$$

The amplification matrix of the implicit multistep scheme is:

$$R_{\mathrm{MS,imp}} = \begin{pmatrix} I_{m_1} & S_1^0 & 0 & \cdots & 0 \\ S_2^0 & I_{m_2} & 0 & \cdots & 0 \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & & \ddots & I_{m_1} & 0 \\ 0 & \cdots & \cdots & 0 & I_{m_2} \end{pmatrix}^{-1} \cdot \begin{pmatrix} e^{Z_1} & S_1^1 & 0 & S_1^2 & \cdots & 0 & S_1^k \\ S_2^1 & e^{Z_2} & S_2^2 & 0 & \cdots & S_1^k & 0 \\ I_{m_1} & 0 & \cdots & \cdots & \cdots & 0 & 0 \\ 0 & I_{m_2} & \ddots & & & \vdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & & \ddots & I_{m_1} & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & 0 & I_{m_2} & 0 & 0 \end{pmatrix} \in \mathbf{C}^{mk \times mk} \tag{19}$$

where S_1^j and S_2^j are defined as previously, but with the Lagrange basis polynomial at points $\tau_0 = -k+1, \tau_1 = -k+2, \ldots, \tau_k = 1$.

In Figures 2(a) and 2(b), we show as an example the evolution of the spectral radius of $R_{\mathrm{MS,exp}}$ and $R_{\mathrm{MS,imp}}$ as a function of Δt for the coupled system from Section 3.2. It is clear that the explicit scheme may be unstable, and the onset of stability matches the divergence of the error in Figure 1(a). The implicit scheme does not suffer from instabilities in the range of time steps considered.



(a) Spectral radius of the explicit scheme

(b) Spectral radius of the explicit scheme

Figure 2. Spectral radii of the recurrence matrices for the multistep scheme applied to the 2×2 coupled test equation with the same parameters as in Figure 1.

4.3. *Other schemes from the literature and their amplification matrices*

It is instructive to compare the stability of our multistep coupling scheme to that of other coupling schemes found in the literature. In particular, we focus on splitting schemes and ImEx ARK methods already discussed in the introduction, and derive the associated recurrence matrices.

4.3.1. *Splitting schemes*

In numerous works, the coupling is implemented in a pragmatic manner, where each solver is run sequentially. This is sometimes referred to as a Gauss–Seidel or sequential splitting, or Conventional Serial Staggered (CSS) scheme for instance in the fluid-structure community [1].

This actually corresponds to a first-order Lie splitting of the original coupled problem, where equation (14) is decomposed as:

$$\mathrm{d}_t y = \underbrace{\begin{pmatrix} A_1 & A_{12} \\ 0 & 0 \end{pmatrix}}_{\equiv B} y + \underbrace{\begin{pmatrix} 0 & 0 \\ A_{21} & A_2 \end{pmatrix}}_{\equiv C} y. \tag{20}$$

As for the multistep coupling scheme, the study of stability is conducted considering an exact integration of each subsystem, i.e. only splitting errors are considered. We will use the first-order (Lie) and second-order (Strang) splittings [34], which still are extensively used in a wide variety of applications [9]. The corresponding amplification matrices are:

$$R_{\mathrm{Lie}} = \exp(\Delta t C)\exp(\Delta t B), \tag{21}$$

$$R_{\mathrm{Strang}} = \exp\left(\tfrac{1}{2}\Delta t C\right)\exp(\Delta t B)\exp\left(\tfrac{1}{2}\Delta t C\right). \tag{22}$$

There also exists another version of each splitting, in which the operator C is integrated before B. However, we observed that these schemes behaved identically in terms of stability. Higher-order splitting schemes can only be obtained if complex or negative time steps are used, which is usually impossible with existing solvers, and even more so when applied to dissipative problems. Note that the Strang splitting has been independently "reinvented" in the fluid-structure as the Improved Serial Staggered (ISS) scheme [1].

It should be stressed that dynamic adaptation of the coupling time step (splitting time step) can only be done with a large additional cost [35].

4.3.2. *ImEx ARK schemes*

In the context of ImEx ARK schemes, the operator A from equation (14) is decomposed as follows:

$$\mathrm{d}_t y = \underbrace{\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}}_{\equiv M} y + \underbrace{\begin{pmatrix} 0 & A_{12} \\ A_{21} & 0 \end{pmatrix}}_{\equiv N} y$$

i.e. the internal dynamics are integrated implicitly, while the coupling terms are treated explicitly. Codes that already provide an optimised algorithm (e.g. Newton with preconditioned linear solver) for an implicit scheme can thus be coupled via an ImEx scheme and reuse this algorithm to solve each stage of the implicit part of the ImEx scheme. Such a scheme may also offer time adaptation capabilities [27] with low-cost error estimates.

In the most general case, such methods are built to solve the following type of problem [36]: $\mathrm{d}_t y = \sum_{\nu=1}^{N} F^{(\nu)}(y)$ where $y \in \mathbb{R}^m$, $m \in \mathbb{N}^*$. At the n-th time step, for any $y_n \in \mathbb{R}^m$, an s-stage ARK scheme is defined by the Butcher tableaus $A^{(\nu)} = (a_{ij}^{(\nu)})_{1\le i,j\le s}$, $b^{(\nu)} = (b_i^{(\nu)})_{1\le i\le s}$ and $c^{(\nu)} = (c_i^{(\nu)})_{1\le i\le s}$. A single step is given by:

$$\begin{cases} \underline{Y}_i = y_n + \Delta t \sum_{\nu=1}^{N}\sum_{j=1}^{s} a_{ij}^{(\nu)} F^{(\nu)}(\underline{Y}_j) \\ y_{n+1} = y_n + \Delta t \sum_{\nu=1}^{N}\sum_{i=1}^{s} b_i^{(\nu)} F^{(\nu)}(\underline{Y}_i). \end{cases} \tag{23}$$

The comparisons conducted in this paper are restricted to the case $N = 2$ operators. These schemes are called ARK_2 schemes. Therefore, operators M and N given above correspond to $F^{(1)}(y) = My$ and $F^{(2)}(y) = Ny$. The concatenation of the vectors $\underline{Y}_i$ is denoted $\underline{Y} = (\underline{Y}_1^t, \ldots, \underline{Y}_s^t)^t$. For clarity we use the Kronecker product $\otimes$ and the column matrix $\mathbb{1}_s \in \mathbb{R}^{s\times 1}$ filled with ones. Following equation (23):

$$\underline{Y} = \underbrace{\Delta t\left(A^{(1)} \otimes M + A^{(2)} \otimes N\right)}_{\text{dimension } ms \times ms} \underline{Y} + \mathbb{1}_s \otimes y_n.$$

This leads to the following result:

$$\underline{Y} = \left[I_{ms} - \Delta t\left(A^{(1)} \otimes M + A^{(2)} \otimes N\right)\right]^{-1} (\mathbb{1}_s \otimes I_m) y_n. \tag{24}$$

Equation (23) can be rewritten as $y_{n+1} = y_n + \Delta t\big(b^{(1)} \otimes M + b^{(2)} \otimes N\big)\underline{Y}$, giving the expression of the recurrence matrix for ARK_2 schemes:

$$y_{n+1} = \underbrace{\Big\{I_m + \Delta t\big(b^{(1)} \otimes M + b^{(2)} \otimes N\big)\Big[I_{ms} - \Delta t\big(A^{(1)} \otimes M + A^{(2)} \otimes N\big)\Big]^{-1}(\mathbb{1}_s \otimes I_m)\Big\}}_{\equiv R_{\mathrm{ARK}_2}} y_n. \tag{25}$$

The multistep coupling scheme will be compared to some state-of-the-art ImEx ARK_2 schemes. The chosen ARK_2 schemes are of orders 3 to 5. They integrate the block-diagonal operator M (representing the subsystems internal dynamics) with an L-stable, stiffly-accurate, singly diagonally implicit Runge–Kutta method with an explicit first stage (ESDIRK), while they integrate the coupling operator N with an explicit Runge–Kutta method (ERK). The third-order scheme (ARK3(2)4L[2]SA) is taken from [10] and the fourth- and fifth-order schemes (ARK4(3)7L[2]SA_1 and ARK5(4)8L[2]SA_2) are found in [27]. A simple first-order ImEx method combining the implicit and explicit Euler schemes is also considered.

4.4. *Stability analysis for the asymmetrical case*

In this section, we study the stability of the multistep coupling scheme on the 2×2 coupled test equation (11) using the first approach suggested in Section 4.

The parameter $\gamma = \frac{a_{12}a_{21}}{a_1 a_2}$ is considered to be a measure of the strength of the coupling [33], while δ is the ratio of the internal characteristic times of both subsystems. Figure 3 shows the stability limits for the various methods. Each curve denotes the lower limit of the stability domain. The continuous system is only stable for $\gamma < 1$, which is used as the upper limit for the vertical axis. We do not consider the zone $z_1 = a_1\Delta t > 0$ which corresponds to an unstable internal dynamics for subsystem 1. The stability limits of the implicit multistep coupling schemes of order 1 to 4 do not appear in these figures, since they are much wider than those of the other methods.

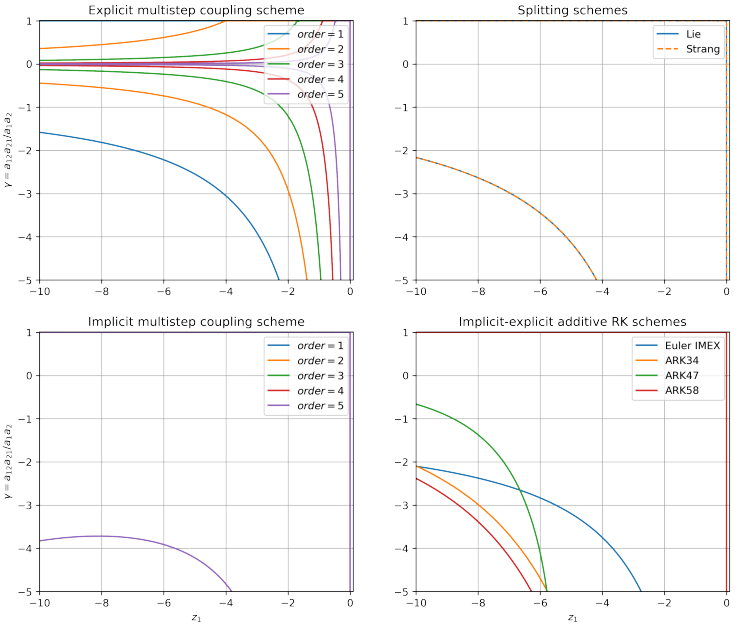
For a given scheme and a given set $\{z_1, \delta\}$, there exist $1 \geqslant \gamma_+(z_1,\delta) > 0 > \gamma_-(z_1,\delta)$ such that the scheme is stable in the range $\gamma \in [\gamma_-, \gamma_+]$. The most stable schemes have $\gamma^+ = 1$. The case $\gamma = 0$, which corresponds to a one-way coupling or to the absence of coupling, is stable for all schemes for any δ and z_1. It can also be seen that the width $\gamma_+ - \gamma_-$ of the stability domain tends to ∞ as z_1 tends to 0.

The implicit multistep scheme has a large stability domain even for high orders. The first-order explicit multistep scheme is comparable to the splittings and the first-order ImEx scheme, however higher-order variants are much less stable. Both splitting schemes have the same stability domain which is competitive with the other explicit or ImEx methods. ImEx schemes show good stability properties when the δ is small and when the internal dynamics is nonstiff (z_1 small). When the internal stiffness increases as $|z_1|$ is increased, the stable range of γ is reduced. For a large value of δ, $z_2 = \delta z_1$ becomes a very stiff term and causes a large diminution of the stability domain for ImEx schemes. Note that, in Figure 3(b) for large values of γ, e.g. -5, and low values of $|z_1|$, the stability of the ImEx ARK schemes can be comparable or better than the implicit multistep scheme, and much more favourable than the explicit variant.

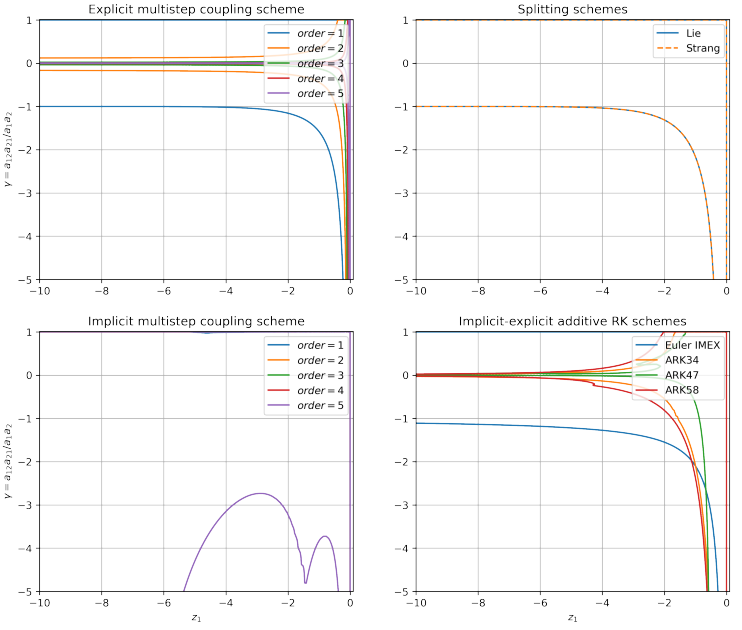
4.5. *Stability analysis for the symmetrical case*

We now turn to the symmetrical model from Section 4.1.2, which enables us to study the effect of imaginary components in the coupling terms.

Figure 4 shows the stability limit of the studied coupling schemes for two values of the diagonal term $z_1 = a_1\Delta t$. The stability limit is thus a function of the real and imaginary parts of the



(a) $\delta = 0.1$



(b) $\delta = 10$

Figure 3. Stability comparison between the implicit and explicit multistep methods, the Lie and Strang splittings and some ImEx schemes (asymmetrical case).

coupling terms $z_{12} = z_{21}$, and is here normalised by $|z_1|$ to help with the visual comparison. The interior of the stability domain is simply found by noting that it contains the origin $z_{12} = 0$. The stability limit is symmetrical with respect to both the imaginary and real parts of z_{12}, thus only the positive quarter-plane is shown. The red dotted line $\mathrm{Re}(z_{12}) = |z_1|$ indicates the stability limit of the continuous system.

We may first observe that the stability domains tend to shrink as the order increases, with some exception for the ImEx schemes. The implicit formulation is more stable than the explicit one and some of its stability domains are unbounded for the lowest orders. For both tested values of z_1, the implicit stability domains cover the whole stability domain of the continuous system along the real axis at all orders, while the explicit formulation only verifies this property for all z_1 at order 1, or at higher-orders but for smaller values of z_1 not shown here. This behaviour for purely real coefficients is coherent with the results from the previous section, where it corresponds to the case $\gamma \in [0, 1]$.

In the case of the explicit multistep and splitting schemes, the stiffer the internal dynamics is (large negative z_1), the smaller the stability domain is in the figure, i.e. relative to $|z_1|$, which is coherent with our observations in the asymmetrical real case in Section 4.4. However, in the absolute space (without the scaling $1/|z_1|$) the stability domains are actually growing. The explicit multistep coupling scheme is comparable to the splitting schemes in stability only at order 1, and its stability domain is close to that of the first-order ImEx method.

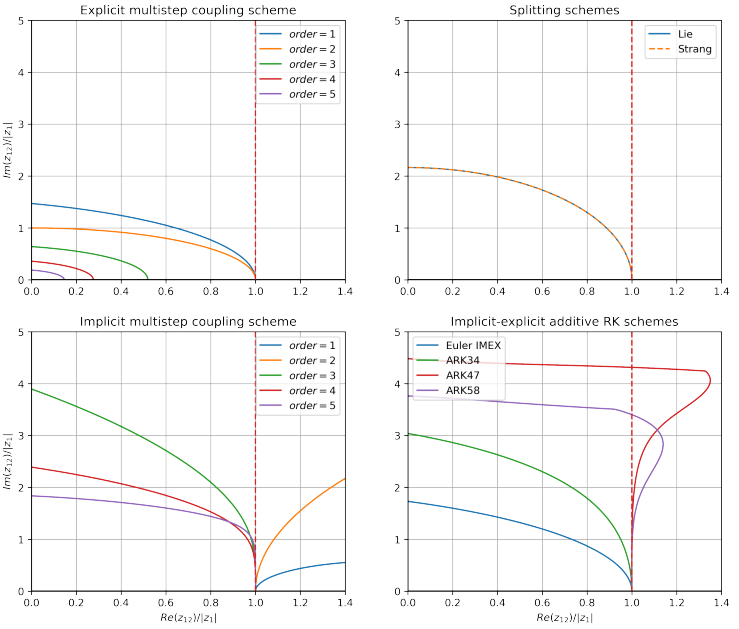
At fixed order, the implicit stability domains grow when the internal dynamics of the subsystems becomes stiffer (larger $|z_1|$). In Figure 4(b), it can even be seen that the implicit stability domains of order 3 to 5 have become unbounded and cover the whole continuous stability domain $\mathrm{Re}(z_{12}) < |z_1|$, except for a small portion near the point $\mathrm{Im}(z_{12}) = 0$, $\mathrm{Re}(z_{12}) = |z_1|$ for higher-order variants. On the opposite, ImEx, explicit multistep coupling and splitting schemes always have bounded stability domains. This observation suggests that a notion of A-stability, adapted from that for ODE integrators, could be defined, requiring that a scheme be stable over the whole stability domain of the continuous system.

4.6. *Discussion*

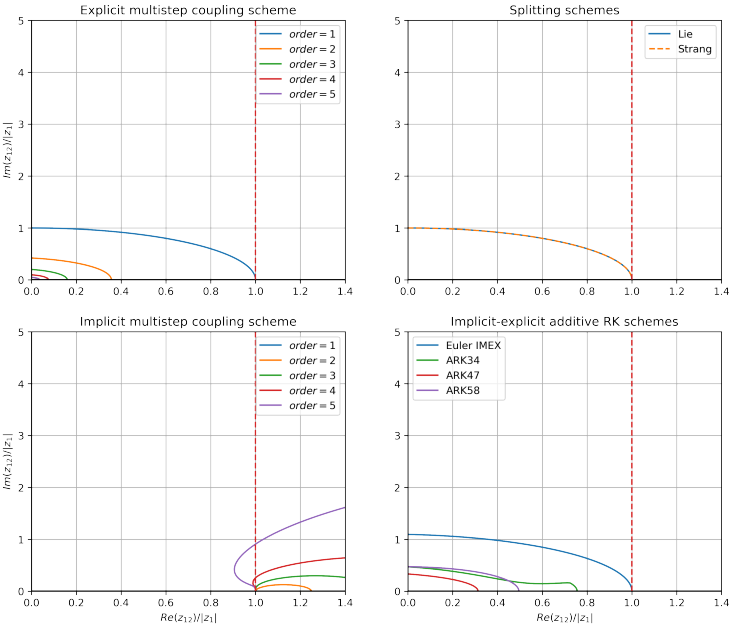
Both previous sections have shown that the stability domains may vary strongly between the different schemes and also depending on the choice of parameters. The choice of a coupling scheme should be driven by multiple aspects: stability, efficiency, ease of use, ease of implementation. In this section, we discuss each scheme and indicate situations where the multistep coupling scheme is a valuable choice.

The Lie splitting scheme is the simplest scheme available, as it is the easiest to implement, requiring only a sequential call to each solver in turn (Gauss–Seidel coupling). It is slightly more stable than the first-order explicit multistep scheme, where the solvers can however be called in parallel (Jacobi coupling), which may greatly improve the computational efficiency. The Strang splitting is an extension to order 2 which has no overcost, since it is essentially a shifted variant of the Lie splitting. It retains the same stability domain while improving accuracy, and should thus be a scheme of choice for applications requiring simple coupling algorithms.

Higher-order explicit multistep schemes can be seen as high-order generalisation of parallel (Jacobi) coupling. The stability domain is reduced, however the accuracy is greatly increased. Note that the explicit multistep coupling schemes provide natural error estimates to guarantee a certain level of accuracy on the approximation of the coupling variables [18,19]. This is a strong advantage compared to splitting schemes, which do not offer time adaptation natively, or only at a large additional cost [35]. Indeed, this allows a user to directly specify an error tolerance,



(a) $z_1 = -1$



(b) $z_1 = -10$

Figure 4. Stability comparison between the implicit and explicit multistep methods, the Lie and Strang splittings and some ImEx schemes (symmetrical case). The red dotted line $\mathrm{Re}(z_{12}) = |z_1|$ indicates the stability limit of the continuous system.

without having to try different coupling time steps to assess the temporal convergence of a simulation.

The chosen ImEx schemes perform well in terms of stability, and have a stability up to 5 times larger than the explicit multistep coupling scheme for the higher-order variants. The next section will somewhat mitigate this observation for certain configurations commonly encountered in practice. ImEx schemes are however closer in their principle to monolithic methods. Indeed, every subsystem must evolve with a single time step, i.e. no substepping is possible, which makes it difficult to accurately and efficiently couple systems with very different time scales. Furthermore, they require that each solver possesses a nonlinear system solution algorithm (Newton or similar) for the solution of the implicit part of the scheme. Thus, solvers only using explicit time integrators internally cannot be used, and solvers possessing an implicit scheme must however be heavily modified to permit a synchronised temporal advancement of all solvers at each stage of the ImEx scheme. Moreover, the implicit Runge–Kutta method used in the ImEx scheme may not be suited to all physics considered in a given multiphysics problem.

The implicit multistep scheme generally offers the best stability of all studied scheme, and is the only one to offer unbounded stability domains in certain configurations. This however comes at the cost of having to solve a potentially nonlinear system on the coupling variables at each step. For nonstiff couplings, a simple relaxation procedure can suffice. If the coupling is stronger, Newton-like methods may be needed. For surface-coupled systems, interface quasi-Newton method have emerged recently [28], where a Jacobian of the coupling residuals is iteratively approximated from initial relaxation iterations. These methods have proved valuable for many applications, in particular fluid-structure interaction. However, there is still much research to be conducted on the topic of efficient solution algorithms for the implicit coupling.

The multistep coupling scheme overall has acceptable stability limits in its explicit variant, and much better ones in its implicit form. It seems to be a valuable choice for many applications, due to its ease of implementation. Indeed, solvers can still use their existing physics-optimised time integrator and are free to subcycle. Modifications to be made to the codes are minor, mostly requiring the ability to update the coupling terms via the polynomial predictors $\widehat{u}_i^n(t)$ during the solver's subcycles. Another advantage is that the computational cost remains the same at all orders, similarly to multistep ODE integrators.

The implementation of the high-order coupling procedure in an HPC context is being tackled in the coupling library CWIPI [7], with the aim of providing an easy-to-use interface for large scale and industrial simulations.

5. A conjugate heat transfer test case

The previous section has proposed an analysis of the coupling stability for the simple 2×2 coupled test equation (3). In this section, we consider a more complex coupled model to determine how the stability of each scheme is affected. We study the application of the multistep coupling strategy to a conjugate heat transfer (CHT) test case, with two solid slabs of finite length exchanging heat through an interface. This case is recognized as a representative configuration for multidimensional fluid-solid thermal interactions, in particular with respect to the stability restrictions on the time step. Indeed, the stability limit is usually controlled by the diffusive processes near the coupling surface [37], whose stiffness is linked to the mesh refinement in the direction perpendicular to the coupling surface.

This classical problem [18,19] is briefly recalled for completeness. Figure 5 provides a sketch of the configuration. For simplicity, both slabs have the same length, same space discretisation (with a second-order finite-volume approach), and same thermal properties: $\lambda_1 = \lambda_2 = 1\,\mathrm{W/m/K}$,

$c_1 = c_2 = 1\,\mathrm{J/kg/K}$. They are only different in their densities ρ_1 and ρ_2. We set $\rho_1 = 1\,\mathrm{kg/m}$. Each slab has a length $L = 1\,\mathrm{m}$ and is discretised uniformly with 50 cells.

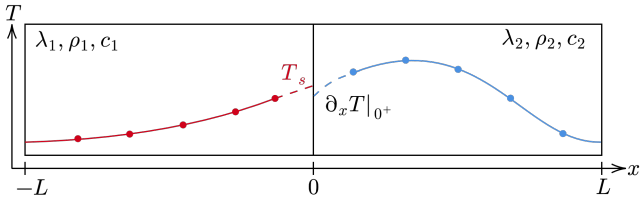
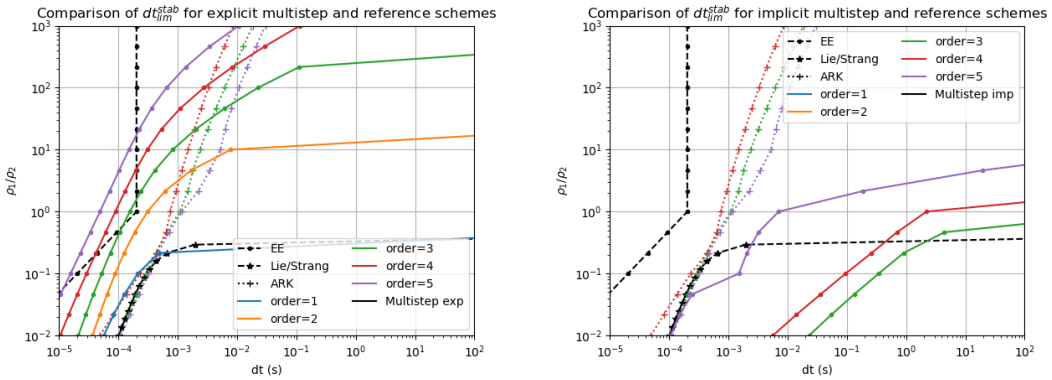


Figure 5. Sketch of the 1D conjugate heat transfer case.

The coupling is of the Neumann–Dirichlet type, with the first slab receiving a heat flux prescribed by the second solver (extrapolated to the interface by a one-sided second-order scheme), and the second slab being imposed an interface temperature prescribed by the first solver in a similar fashion. Taking each slab as a standalone system with a dedicated solver, we obtain a coupled system representative of the CHT between a fluid solver and a heat diffusion solver. Following the same approach as in Section 4.2, we construct for each coupling scheme the recurrence matrix for the CHT problem, with the submatrices A_{ij} corresponding to the terms arising from the finite-volume discretisation [19]. By varying the time step through a dichotomy process, we can determine the corresponding maximum stable time step of a coupling scheme applied to this configuration.

Existing studies on the stability of this problem show for the simple first-order coupling scheme that the ratio of effusivities is the main parameter impacting stability [37], which in our case amounts to the ratio of densities. Thus, we propose to determine the maximum stable time step for each coupling method as a function of the ratio $\rho_1/\rho_2 \in [10^{-2}, 10^{3}]$, using a dichotomy process for each ratio and method to find Δt such that the spectral radius of the associated recurrence matrix is 1.

The results are shown in Figure 6(a) for the explicit multistep coupling scheme, and in Figure 6(b) for the implicit scheme. Both figures include the stability limits of the ImEx schemes and splitting schemes from Section 4.3. We also include the stability limit of the explicit Euler scheme, which indicates how a standard monolithic explicit scheme would perform.



(a) Explicit formulation

(b) Implicit formulation

Figure 6. Maximum stable time steps for the 1D CHT case.

It can be noted that the stability limits obtained for the ImEx schemes and the explicit multistep coupling schemes closely reproduce those obtained in [18], where the Python demonstrator RHAPSOPY [38] of the multistep coupling has been used to perform several fixed-time-step integrations, and stability limits have been found based on a divergence criterion for the obtained solutions. For the implicit multistep scheme however, high-order variants have obtained here a lower stability limit. This may be attributed to the low-order initialisation (first steps) of the simulations from the previous work, which may improve the perceived numerical stability of the method.

Both splittings have the same stability limit, which is virtually infinite for density ratios above 0.5. We observe that the explicit multistep coupling scheme is competitive for large values of ρ_1/ρ_2, and that an increase in order yields a decrease in stability, as seen with the test equation from Section 4.5. ImEx schemes have a relatively low stability limit which is only advantageous for intermediate density ratios. This could indicate that the explicit multistep scheme has a lower stiffness leakage [10] than the chosen ImEx schemes, i.e. stiffness from the internal dynamics of the subsystems may leak more into the explicit part under certain conditions, causing stronger stability restrictions.

The implicit scheme has, for all orders, a much larger stability limit, even virtually unlimited for schemes of order up to 2, or at all orders for the higher density ratios. This clearly demonstrate the advantage of the implicit formula. Note that the interface boundary conditions should be reversed when $\rho_1 < \rho_2$ [37], hence the implicit scheme can be seen to remain robust even when the boundary conditions choice is inadequate and favours instability.

For most gas-thermal coupling, the density ratio is above 100, hence the explicit multistep scheme or the Strang splitting would be a good choice, owing to their simplicity of use and implementation. For stronger coupling where the density ratio is closer to 1, e.g. liquid-solid thermal coupling, the multistep implicit coupling scheme would be preferable. The splitting schemes may also be advantageous due to their explicit nature. A study on the achieved precision should be conducted to discriminate between these methods.

This case, which is representative of a category of realistic multiphysics applications where diffusion dominates the coupling process, shows that the multistep coupling scheme in its implicit variant has a very good stability. The explicit variant can be competitive for intermediate density ratios, and very stable for large ratios. The chosen ImEx scheme, although they are stable for larger time steps than a standard monolithic explicit scheme, are only relevant for intermediate values of the density ratio.

Note that, in the context of adaptive time stepping based on error estimates, the coupling time step will remain within or at the border of the stability region, thus preventing instability. However, for a poorly stable scheme, the time step may end up being dictated by the stability limit instead of the accuracy requirement, greatly reducing the computational efficiency [25, Section IV.2]. Therefore, stability considerations may be helpful in diagnosing such an issue and in choosing a well-suited coupling scheme.

The conjugate heat transfer highlights the strong influence of physical parameters on the stability of the schemes. It would be interesting to search for a link between this complex coupled system of PDEs and the simple 2×2 coupled test equation (see Section 3.1). We are currently investigating whether a set of 2×2 test equations can be generated from the complete CHT system, in such a way that each of these simpler models could be located within the stability domains explored in Section 4 to assess the stability of the original CHT system. Such an approach would have the capability of predicting stability a priori, offering a very efficient way to choose the proper time step.

6. Conclusion

In this work, a thorough analysis of the essential convergence properties of the multistep coupling scheme has been presented, extending results initiated in [18]. Inspired by Dahlquist's test equation for the stability analysis of ODEs, a 2×2 coupled test equation has been proposed. Relying on a recurrence matrix formulation of the multistep coupling scheme, the stability limit can be studied systematically by simple considerations on the spectral radius of the matrix. The analysis shows that the multistep scheme stability is good compared to other schemes, but is reduced as the polynomial order is increased. Implicit coupling provides much greater stability than the explicit variant. State-of-the-art ARK ImEx methods show good stability limits, surpassing the stability of the multistep coupling scheme at higher orders, however at the cost of being closer to a monolithic integration scheme, thus more demanding in terms of implementation, while prohibiting subcycling. Low-order Lie and Strang splittings show a very good performance in terms of stability, thus providing interesting coupling schemes for applications less affected by coupling accuracy. Application on a 1D conjugate heat transfer test case shows that the tendencies obtained with the 2×2 coupled test equation convey into the analysis of a system representative of actual multiphysics applications.

Overall, the multistep coupling scheme is a valuable coupling scheme. It forms a natural extension of the basic parallel coupling schemes often used in practice, only requiring minor modifications in the codes to be used. The order of convergence in time can be arbitrary, and an increase in order does not result in an increase in computational cost, as it only requires storing more past values of the coupling variables. Although not discussed in this article, this scheme allows for error estimates to be simply derived from the polynomials, enabling a dynamic adaptation of the coupling time step to guarantee accuracy and stability, while also optimising the computational cost.

The multistep coupling method is the subject of ongoing work. Regarding the stability analysis, a quantitative link should be made between the 2×2 coupled test equation (14) and actual coupled partial differential equations, so as to allow an a priori analysis of the coupling stability. The multistep scheme is being implemented in the open-source HPC library CWIPI [6,7] with the aim of being used in a wide range of applications. Such a library will provide an easy-to-use package to easily incorporate a high-order adaptive multistep coupling scheme into new or existing coupled applications. Finally, the optimisation of the computational cost of the implicit variant will be a subject of specific work, in particular regarding the application of specific Newton-like algorithms for the solution of the nonlinear system obtained at each step.

Appendix A. Local truncation error constants

Let us consider the linear problem from equation (14). It can be reformulated as:

$$\mathrm{d}_t y = A_{\mathrm{int}} y + A_{\mathrm{cpl}} y \tag{26}$$

with the internal dynamics operator $A_{\mathrm{int}} \equiv \left(\begin{smallmatrix} A_1 & 0 \\ 0 & A_2 \end{smallmatrix}\right)$ and the coupling operator $A_{\mathrm{cpl}} \equiv \left(\begin{smallmatrix} 0 & A_{12} \\ A_{21} & 0 \end{smallmatrix}\right)$. This corresponds to the case $y = (y_1^t, y_2^t)^t$, $u_1 = y_2$ and $u_2 = y_1$.

Over a coupling time step $[t_n, t_{n+1}]$, the multistep scheme corresponds to the integration of the following modified equation:

$$\mathrm{d}_t y = A_{\mathrm{int}} y + A_{\mathrm{cpl}} \widehat{y}_n \tag{27}$$

where $\widehat{y}_n$ is the polynomial approximation of y:

$$\widehat{y}_n(t) = \sum_{j=\delta k}^{k+\delta k} y_{n-j} \prod_{\substack{l=\delta k \\ l \neq j}}^{k+\delta k} \frac{t - t_{n-l}}{t_{n-j} - t_{n-l}} \tag{28}$$

with k the degree of the interpolation polynomial, and $\delta k = 0$ in the explicit case, -1 in the implicit case (so that t_{n+1} is a sampling point of the interpolant).

The exact solution y_{exa} and the multistep solution y_{n+1} read:

$$\begin{aligned} y_{\text{exa}}(t_{n+1}) &= e^{A_{\text{int}}\Delta t} y_{\text{exa}}(t_n) + \int_0^1 e^{A_{\text{int}}\Delta t(1-\eta)} A_{\text{cpl}} \Delta t\, y_{\text{exa}}(t_n + \eta\Delta t)\, \mathrm{d}\eta \\ y_{n+1} &= e^{A_{\text{int}}\Delta t} y_n + \int_0^1 e^{A_{\text{int}}\Delta t(1-\eta)} A_{\text{cpl}} \Delta t\, \widehat{y}^n(t_n + \eta\Delta t)\, \mathrm{d}\eta \end{aligned} \tag{29}$$

with $\Delta t = t_{n+1} - t_n$.

The local truncation error can be obtained by applying the multistep coupling scheme to the exact solution, i.e. using the exact solution to generate the interpolation polynomial $\widehat{y}_n$ and to set the initial condition $y_n = y_{\text{exa}}(t_n)$. Subtracting both solutions yields the local truncation error $\delta y_{n+1} \equiv y_{n+1} - y_{\text{exa}}(t_{n+1})$ for a single coupling step:

$$\delta y_{n+1} = \int_0^1 e^{A_{\text{int}}\Delta t(1-\eta)} A_{\text{cpl}} \Delta t \underbrace{[y_{\text{exa}} - \widehat{y}_n](t_n + \eta\Delta t)}_{\equiv \epsilon} \mathrm{d}\eta. \tag{30}$$

Since $\widehat{y}_n$ interpolates the exact solution, its interpolation error can be expressed as:

$$\epsilon(t_n + \eta\Delta t) = \frac{\Delta t^{k+1} y_{\text{exa}}^{(k+1)}(t_n)}{(k+1)!} \prod_{j=\delta k}^{k+\delta k} (\eta + j) + \mathcal{O}(\Delta t^{k+2}). \tag{31}$$

Then, the local truncation error reads:

$$\delta y_{n+1} = \int_0^1 e^{A_{\text{int}}\Delta t(1-\eta)} A_{\text{cpl}} \left[\frac{\Delta t^{k+2} y_{\text{exa}}^{(k+1)}(t_n)}{(k+1)!} \prod_{j=\delta k}^{k+\delta k} (\eta + j) + \mathcal{O}(\Delta t^{k+3}) \right] \mathrm{d}\eta. \tag{32}$$

Since we are interested in the asymptotic convergence regime where Δt is small, we can expand the exponential as $e^{A_{\text{int}}\Delta t(1-\eta)} = I + A_{\text{int}}\Delta t(1-\eta) + \mathcal{O}(\Delta t^2)$. The first-order term $A_{\text{int}}\Delta t(1-\eta)$ multiplied by the polynomial of η produces a contribution of order $k+3$ that is absorbed by the $\mathcal{O}(\Delta t^{k+3})$ term, which allows for the following simplified form:

$$\delta y_{n+1} = \Delta t^{k+2} \underbrace{\cdot \frac{1}{(k+1)!} \cdot \int_0^1 \prod_{j=\delta k}^{k+\delta k} (\eta + j)\, \mathrm{d}\eta}_{\text{error constant}} \cdot A_{\text{cpl}} \cdot y_{\text{exa}}^{(k+1)}(t_n) + \mathcal{O}(\Delta t^{k+3}). \tag{33}$$

This simplification is coherent with the fact that the internal dynamics is assumed to be integrated exactly, so that its accuracy can only be indirectly worsened due to the polynomial approximation of the coupling term. In particular, when there is no coupling, $A_{\text{cpl}} = 0$ and the error is 0.

The error constants exactly correspond to those of the Adams–Bashforth multistep schemes in the explicit coupling case, and to those of the Adams–Moulton methods in the implicit case [29, Section 5.1.1]. This is coherent, since the simplification following the series expansion has the same effect as setting $A_{\text{int}} = 0$, in which case the multistep coupling scheme reduces to an Adams multistep scheme. Note that, in this particular setting, $y'_{\text{exa}} = A_{\text{cpl}} y$, therefore the error can be reformulated in the more traditional linear multistep form [29, equation (5.2)]:

$$\delta y^{n+1} = \frac{1}{(k+1)!} \cdot \int_0^1 \prod_{j=\delta k}^{k+\delta k} (\eta + j)\, \mathrm{d}\eta \cdot \Delta t^{k+2} \cdot y_{\text{exa}}^{(k+2)}(t_n) + \mathcal{O}(\Delta t^{k+3}) \tag{34}$$

where the operators prescribing the system dynamics no longer appear.

The predicted ratios of error constants between the implicit and explicit multistep coupling schemes of the same order are listed in Table 1, alongside the numerically evaluated ratios based on fits of the asymptotic parts of the convergence curves from Figure 1. The ratios indicate that the implicit coupling scheme is increasingly more accurate than its explicit counterpart as the

order is increased. This can be understood by the behaviour of the error polynomial (main term of ϵ), which diverges monotonically on $[t_n, t_{n+1}]$ in the extrapolation case, while it is bounded on that interval in the interpolation case (implicit coupling).

Table 1. Ratio of the implicit error constant divided by the explicit error constant, as predicted by the theory, and as evaluated numerically from Figure 1.

Order	Theory	Figure 1
1	1	1.0
2	5	5.2
3	9	9.4
4	13.2	13.5
5	17.6	17.7
6	22.1	22.8

Acknowledgments

The manuscript was written through contributions of all authors. All authors have given approval to the final version of the manuscript.

The authors would like to thank Josselin Massot and Nicole Spillane (CMAP) for the reading of the draft of the paper and fruitful discussions.

Declaration of interests

The authors do not work for, advise, own shares in, or receive funds from any organization that could benefit from this article, and have declared no affiliations other than their research organizations.

References

[1] C. Farhat and M. Lesoinne, "Two efficient staggered algorithms for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems", *Comput. Methods Appl. Mech. Eng.* **182** (2000), no. 3, pp. 499–515.

[2] A. Bourlet, F. Tholin, J. Labaune, F. Pechereau, A. Vincent-Randonnier and C. O. Laux, "Numerical model of restrikes in gliding arc discharges", *Plasma Sources Sci. Technol.* **33** (2024), no. 1, article no. 015010.

[3] V. Kazemi-Kamyab, A. H. Van Zuijlen and H. Bijl, "A high order time-accurate loosely-coupled solution algorithm for unsteady conjugate heat transfer problems", *Comput. Methods Appl. Mech. Eng.* **264** (2013), pp. 205–217.

[4] L. Boulet, P. Bénard, G. Lartigue, V. Moureau, S. Didorally, N. Chauvet and F. Duchaine, "Modeling of conjugate heat transfer in a kerosene/air spray flame used for aeronautical fire resistance tests", *Flow Turbul. Combust.* **101** (2018), pp. 579–602.

[5] A. Langenais, F. Vuillot, J. Troyes and C. Bailly, "Accurate simulation of the noise generated by a hot supersonic jet including turbulence tripping and nonlinear acoustic propagation", *Phys. Fluids* **31** (2019), article no. 016105 (31 pages).

[6] A. Refloch, B. Courbet, A. Murrone, et al., "CEDRE software", *AerospaceLab J.* (2011), no. 2, pp. 1–10.

[7] ONERA, *onera/cwipi: Library for coupling parallel scientific codes via MPI communications to perform multi-physics simulations*. Online at https://github.com/onera/cwipi (accessed on October 29, 2025).

[8] A. Grenouilloux, R. Letournel, N. Dellinger, K. Bioche and V. Moureau, "Large-eddy simulation of solid/fluid heat and mass transfer applied to the thermal degradation of composite material", in *Proceedings of the 14th Direct and Large Eddy Simulation (DLES) Workshop*, ERCOFTAC, 2024.

[9] D. E. Keyes, L. C. McInnes, C. S. Woodward, et al., "Multiphysics simulations: challenges and opportunities", *Int. J. High Perform. Comput. Appl.* **27** (2013), no. 1, pp. 4–83.

[10] C. A. Kennedy and M. H. Carpenter, "Additive Runge–Kutta schemes for convection-diffusion-reaction equations", *Appl. Numer. Math.* **44** (2003), no. 1, pp. 139–181.
[11] M. Günther, A. Kværnø and P. Rentrop, "Multirate Partitioned Runge–Kutta Methods", *BIT Numer. Math.* **41** (2001), no. 3, pp. 504–514.
[12] C. W. Gear, "Automatic multirate methods for ordinary differential equations", in *Information processing 80* (S. H. Lavington, ed.), IFIP Congress Series, vol. 8, North-Holland, 1980, pp. 717–722.
[13] C. W. Gear and D. R. Wells, "Multirate linear multistep methods", *BIT* **24** (1984), no. 4, pp. 484–502.
[14] S. Kang, A. Dener, A. Hamilton, H. Zhang, E. M. Constantinescu and R. L. Jacob, "Multirate partitioned Runge–Kutta methods for coupled Navier–Stokes equations", *Comput. Fluids* **264** (2023), article no. 105964.
[15] J. Wensch, O. Knoth and A. Galant, "Multirate infinitesimal step methods for atmospheric flow simulation", *BIT Numer. Math.* **49** (2009), pp. 449–473.
[16] J. J. Loffeld, A. Nonaka, D. R. Reynolds, D. J. Gardner and C. S. Woodward, "Performance of explicit and IMEX MRI multirate methods on complex reactive flow problems within modern parallel adaptive structured grid frameworks", *Int. J. High Perform. Comput. Appl.* **38** (2024), no. 4, pp. 263–281.
[17] B. Rüth, B. Uekermann, M. Mehl, P. Birken, A. Monge and H.-J. Bungartz, "Quasi-Newton waveform iteration for partitioned surface-coupled multiphysics applications", *Int. J. Numer. Methods Eng.* **122** (2021), no. 19, pp. 5236–5257.
[18] L. François and M. Massot, "Multistep interface coupling for high-order adaptive black-box multiphysics simulations", in *X International Conference on Computational Methods for Coupled Problems in Science and Engineering* (M. Papadrakakis, B. Schrefler and E. Oñate, eds.), 2023.
[19] L. François, *Multiphysics modelling and simulation of solid rocket motor ignition*, PhD thesis, Institut Polytechnique de Paris (France), 2022.
[20] A. Asad, R. de Loubens, L. François and M. Massot, "High-order adaptive multi-domain time integration scheme for microscale lithium-ion batteries simulations", *SMAI J. Comput. Math.* **11** (2025), pp. 369–404.
[21] R. Kübler and W. Schiehlen, "Two methods of simulator coupling", *Math. Comput. Model. Dyn. Syst.* **6** (2000), no. 2, pp. 93–113.
[22] M. Busch, *Zur effizienten Kopplung von Simulationsprogrammen*, PhD thesis, Universität Kassel (Germany), 2012.
[23] A. Toselli and O. Widlund, *Domain decomposition methods — Algorithms and theory*, Springer Series in Computational Mathematics, vol. 34, Springer, 2004.
[24] M. Gander, P. Henry and G. Wanner, "Landmarks in the history of iterative methods". Online at https://www.unige.ch/~gander/Preprints/LandmarksPaper.pdf.
[25] E. Hairer and G. Wanner, *Solving ordinary differential equations II. Stiff and differential-algebraic problems*, 2nd edition, Springer Series in Computational Mathematics, Springer, 1996.
[26] G. Strang, "On the construction and comparison of difference schemes", *SIAM J. Numer. Anal.* **5** (1968), no. 3, pp. 506–517.
[27] C. A. Kennedy and M. H. Carpenter, "Higher-order additive Runge–Kutta schemes for ordinary differential equations", *Appl. Numer. Math.* **136** (2019), pp. 183–205.
[28] J. Degroote, "Partitioned simulation of fluid-structure interaction: coupling black-box solvers with quasi-Newton techniques", *Arch. Comput. Methods Eng.* **20** (2013), no. 3, pp. 185–238.
[29] U. M. Ascher and L. R. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*, Society for Industrial and Applied Mathematics, 1998.
[30] M. Schatzman, *Numerical analysis: a mathematical introduction*, Oxford University Press, 2002.
[31] V. Kazemi-Kamyab, A. H. Van Zuijlen and H. Bijl, "Analysis and application of high order implicit Runge–Kutta schemes for unsteady conjugate heat transfer: a strongly-coupled approach", *J. Comput. Phys.* **272** (2014), pp. 471–486.
[32] P. Birken, T. Gleim, D. Kuhl and A. Meister, "Fast solvers for unsteady thermal fluid structure interaction", *Int. J. Numer. Methods Fluids* **79** (2015), no. 1, pp. 16–29.
[33] A. Kværnø, "Stability of multirate Runge–Kutta schemes", *Int. J. Differ. Equ. Appl.* **1A** (2000), no. 1, pp. 97–105.
[34] G. Strang, "On the construction and comparison of difference schemes", *SIAM J. Numer. Anal.* **5** (1968), no. 3, pp. 506–517.
[35] S. Descombes, M. Duarte, T. Dumont, V. Louvet and M. Massot, "Adaptive time splitting method for multi-scale evolutionary partial differential equations", *Confluentes Math.* **3** (2011), no. 3, pp. 413–443.
[36] A. Sandu and M. Günther, "A generalized-structure approach to additive Runge–Kutta methods", *SIAM J. Numer. Anal.* **53** (2015), no. 1, pp. 17–42.
[37] M. B. Giles, "Stability analysis of numerical interface conditions in fluid-structure thermal analysis", *Int. J. Numer. Methods Fluids* **25** (1997), no. 4, pp. 421–436.
[38] L. François, *hpc-maths/Rhapsopy: Demonstrator of high-order adaptive code coupling, implicit or explicit*. Online at https://github.com/hpc-maths/rhapsopy (accessed on October 29, 2025).

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus. Mécanique
2025, Vol. 353, p. 1289-1314
https://doi.org/10.5802/crmeca.336

Research article

ArcNum: an Arcane-based numerical framework used in porous media flow simulation applications

Stéphane de Chaisemartin [*,a], **Sylvain Desroziers** [a,b], **Guillaume Enchéry** [a], **Raphaël Gayno** [a], **Jean-Marc Gratien** [a], **Gilles Grospellier** [c], **Thomas Guignon** [a], **Pascal Havé** [a,d], **Benoît Lelandais** [c], **Alexandre l'Héritier** [c], **Anthony Michel** [a], **Aboul Karim Mohamed El Maarouf** [a], **Valentin Postat** [c], **Xavier Tunc** [a] and **Soleiman Yousef** [a]

[a] IFPEN, 1 et 4 avenue de Bois-Préau, 92852 Rueil-Malmaison Cedex, France

[b] Michelin, 23 Place des Carmes Déchaux, 63040 Clermont Ferrand Cedex 9, France

[c] CEA, DAM, DIF, 91297 Arpajon, France

[d] NUANT, Baarerstrasse 20, 6300 Zug, Suisse

E-mail: stephane.de-chaisemartin@ifpen.fr

Abstract. This article presents ArcNum, a framework built on the Arcane platform, designed to easily and efficiently develop and maintain the numerical core in finite-volume and finite-element applications. This framework first enables the automatic generation of code needed to instantiate and handle complex physical models from a textual description, through its component called GUMP. ArcNum also offers software components to create, register and evaluate a set of physical laws, requiring only the specification of their inputs, outputs, and corresponding mathematical formulations. Finally, the framework includes a component named Contribution, which combines law evaluation with automatic differentiation to assemble linear systems efficiently. The framework ArcNum has been used to develop an open source Arcane-based porous media flow simulation proxy-app, named ShArc. Single and two-phase porous media flow simulations performed with ShArc are presented to complete the framework description. In order to illustrate the ability to use ArcNum for High Performance Computing, massively parallel simulations conducted with ShArc are finally presented.

Keywords. Mathematical software framework, code generation, automatic differentiation, Arcane, proxy-app, multiphase porous media flow simulation, massively parallel computing.

Note. Article submitted by invitation.

Manuscript received 9 January 2025, revised 13 October 2025, accepted 20 October 2025.

1. Introduction

The open source Arcane platform for scientific computing [1] is an advanced framework designed to facilitate the development and execution of complex scientific simulations applications. It provides researchers, scientists, and engineers with a comprehensive set of tools and libraries specifically tailored for high-performance computing (HPC) applications.

* Corresponding author

Arcane started being developed at the CEA-DAM, the Military Applications Division of the French Alternative Energies and Atomic Energy Commission, in the early 2000s, before being co-developed with IFPEN, French Institute for Research and Innovation in Energy, Mobility and Environment, from 2007.

During these twenty years of existence, Arcane has been the foundation of several industrial applications, from laser physics to CO_2 storage or geothermal energy simulations. Today eight industrial simulators are based on this platform, for nearly three million code lines from CEA-DAM and IFPEN.

Starting in the 2020s it has been decided to gradually move toward an open source strategy to foster the collaborative development between CEA-DAM and IFPEN, to make the platform available for the community and to foster potential new collaborations. Since the end of 2021, the platform is fully available on GitHub [2].

Arcane offers services dedicated to the application developers, handling for them all the low level computer science services needed by the application and giving them tools to dynamically build the application with an expandable and flexible option system.

Above the Arcane platform, dedicated to handle data and parallelism, numerical frameworks were developed to share the numerical tools (numerical schemes, (non)linear solvers...) between the applications. The works on linear solvers gave birth to Alien [3], a solver interface library designed to provide an application with a unique entry point to the world of linear solver libraries. On top of these (non)linear systems to solve, it was necessary to supply the applications with a way to simplify the complex writing of the Jacobian assembly arising for example in porous media flow simulations. ArcNum was developed with this objective.

In order to share the use of such Arcane-based numerical frameworks with the community, a numerical demonstrator, ShArc, has been released in open source on GitHub [4]. This Arcane-based proxy-application uses the frameworks Alien and ArcNum to solve porous-media flow problems. ShArc is now a proxy-app of Exa-DI NumPEx project; in this context an installation guide is provided on GitLab [5].

This paper is especially dedicated to the ArcNum framework and is organized in the following way. As an introduction, we first recall the main tools provided by the Arcane framework and highlight recent evolutions of the platform. We then present the ArcNum framework itself, detailing its three main components:

(i) a physical data model generator named GUMP;
(ii) a physical law framework;
(iii) a tool to assemble the Jacobian contributions, associated with automatic differentiation.

Finally, numerical results obtained with the ShArc demonstrator, combining all these tools, are presented for two porous media flow simulation test cases. In addition, a performance study of ShArc on massively parallel configurations is reported.

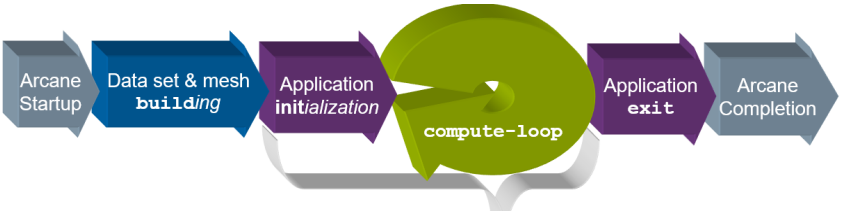


Figure 1. The different stages of the Arcane loop. The init, compute-loop and exit stages are the main stages used by the application developer to plug its *Modules* entry-points.

2. The Arcane Framework

Before diving into the ArcNum tools, we present in this part the Arcane Framework, an open source modular platform for HPC, on which it is based.

2.1. *An open source modular structure*

The Arcane Framework is made of several components, available on a single GitHub repository [6]:

Arccon: made of CMake functions and package finders;
Arccore: containing the base types and parallel tools of Arcane;
Axlstar: gathering the code-generation tools;
Alien: the linear algebra API (cf. Section 2.3);
Arcane: the Arcane platform itself (cf. Section 2.2).

With this modular structure, both Arcane and Alien use the common base provided by Arccore, the set of generators provided by Axlstar and the common CMake tools included in Arccon.

The Arcane framework committee decided in 2021 to release the platform with an open source Apache-2.0 license. All Arcane Framework components are now available on GitHub, see [2], through the *ArcaneFramework* organization, gathering the Arcane developers at CEA-DAM and IFPEN.

2.2. *Arcane: a complete platform for scientific computing*

The C++ platform Arcane provides all the building blocks needed to develop a parallel application. In particular, it provides a mesh structure with easy-to-use parallelization tools, distributed variables and parallel IOs, to name just a few of its features. It is based on an efficient component architecture allowing to define simulation problems as plugins or *Modules*, in the platform language, and to share functionalities through *Services*, defined by an interface, i.e. a class defining a programming contract. We refer to [1] for a more detailed description of the Arcane platform. We here only briefly describe the main functionalities and then present the platform recent developments.

The Arcane framework was originally designed to conduct simulations through the call of various entry-points, implemented by the developer in the *Modules*. Within the *Modules* entry-points, the application developer has access, amongst others, to the simulation mesh, natively distributed over subdomains, the simulation variables, or the simulated case options. The main program of the application only consists in calling the Arcane launcher. The application developer can then focus on developing *Modules*, and possibly *Services* to share treatment between *Modules*. The *Modules* entry-points are registered in a *time-loop*, mainly composed of three stages: the init, compute-loop and exit stages, see Figure 1. Both *Modules* and *Services* are made up of the following elements:

(i) a descriptor file, in xml format, used to defined the entry-points for a *Module* or the interface of a *Service*, and a list of options;
(ii) a C++ class, used to implement the *Module* entry-points and the *Service* interface.

An Arcane mesh is made of the classical geometric items such as cells, faces, nodes and edges. Non-geometric items, dofs (degrees of freedom) or particles, can also be added and connected to the geometric items. The platform can read various formats (vtk, msh and med formats) and generate a few mesh types too (see Section 2.4). It enables the use of several mesh partitioners (Metis, Parmetis, PTScotch and Zoltan) in a transparent manner. The Arcane mesh is intrinsically

unstructured and dynamic, which means that we can add new cells during the simulation and enrich the existing mesh. We can also create dynamically group of items to identify regions of the mesh. The mesh variables are then adapted automatically to take into account these changes in an efficient way. Let us add that a simulation may use several meshes if needed. Distributed variables can be defined on this/these mesh(es). A mesh variable is linked to a specific item family (a family of cells, faces, edges, nodes or dofs) and can have scalar or vector values of various types: Real (Arcane name for double precision floating points), Int16, Int32, Int64...

Arcane is designed to make parallel operations easier. By providing the concept of *SubDomain* with its *own* and *ghost* mesh items, it allows the application developer to easily implement an application based on domain decomposition. Ghost item values can be easily synchronized and explicit communications between the subdomains are not necessary. Arcane also offers load balance functionalities. The communications are classically carried out through mpi. Threaded and hybrid mpi+thread implementations are also available, though not used currently by the main Arcane-based applications.

An Arcane code example is given in Listing 1 to illustrate how mesh, variables and parallel synchronization operations are used to compute a variable defined on the mesh nodes from a second variable defined on the mesh cells. Note that, in this example, the group `mesh->allNodes()` only contains the nodes of the current subdomain. A final synchronization step is therefore required to ensure the correctness of the computation, as some ghost node values may remain inconsistent.

```
1  Arcane::IMesh* mesh; // given by Arcane
2  Arcane::VariableCellInt32 cell_var {VariableBuildInfo{
3                                          mesh,"MyCellVariable"}};
4  Arcane::VariableNodeInt32 node_var {VariableBuildInfo{
5                                          mesh,"MyNodeVariable"}};
6  cell_var.fill(1);
7  node_var.fill(0);
8  ENUMERATE_(Node, inode, mesh->allNodes()) {
9      for (Cell cell : inode->cells()) {
10         node_var[inode] += cell_var[cell];
11     }
12 }
13 node_var.synchronize();
```

Listing 1. Example of the computation of a node variable from a cell variable in parallel.

This abstraction of mesh enumeration and parallel communications allows applications developers to concentrate on the coding of physics and numerical operators. The Arcane level of abstraction remains low with direct access to mesh variables. This allows to easily adapt existing numerical algorithms, and offers the possibility to define finite element or finite volume methods on the top of this Arcane API.

Thanks to this subdomain-based structure, simulators at CEA and IFPEN have been used in parallel simulations on meshes ranging from thousands of millions up to dozens of billion of cells on up to hundreds of thousands of cores, see Section 5.

2.3. *Alien: a generic extensible linear algebra framework*

The Arcane framework now integrates a generic and extensible linear algebra framework called Alien [3]. This linear algebra API was initiated at IFPEN in the 2010s. It was designed to answer the problems arising when using different linear algebra packages in scientific software, and to

easily switch between different sparse matrix storage formats. Alien provides the application developer with a unique API to handle matrices, vectors and to solve linear systems. An example of system assembling and resolution is given in Listing 2. These Alien matrices and vectors are then converted at runtime into the representation of the chosen linear algebra package among PETSc, HYPRE, Trilinos, MTL or SuperLU. Alien also gives access to IFPEN internal solvers such as IFPSolver and MCG Solver [7]. It has been shown in [3] that these data structure conversions don't introduce a significant overhead in CPU-time.

```
auto pm = Env::parallelMng();
auto s = Space{10,"MySpace"};
auto md = MatrixDistribution{s,s,pm};
auto vd = VectorDistribution{s,pm};
auto A = Matrix{md}; // build a 10×10 square matrix
auto x = Vector{vd}; // build a 10 elements vector
auto b = Vector{vd}; // build a 10 elements vector
{
    // fill a tri-diagonal matrix
    auto builder = DirectMatrixBuilder(A,eResetValues);
    builder.reserve(30); // non-zero values
    for (Integer index = 0; index < 10; ++ index) {
        builder(index,index) = 2;
        if (index + 1 < 10) builder(index, index + 1) = -1;
        if (index - 1 >= 0) builder(index, index - 1) = -1;
    }
}
auto* solver = createSolver(/*...*/); // choose your favorite solver
auto status = solver->solve(A,x,b)
```

Listing 2. Handling distributed Alien matrices, vectors and solvers.

2.4. *Handling an always wider variety of mesh*

While initially designed only for standard unstructured meshes, the Arcane framework has been gradually enriched with:

(i) degenerate hexahedral cells,
(ii) generated Cartesian or honeycomb meshes,
(iii) Adaptive Mesh Refinement (AMR), both for unstructured meshes and Cartesian meshes,
(iv) polyhedral meshes,
(v) uniform mesh refinement, to generate a finer mesh.

All these mesh types or features are available in parallel. Degenerate heaxahedral cells, see Figure 2, have been added to handle complex soil geometries, see Figure 3. Such cells can be refined by the Arcane AMR procedure.

A mesh can be loaded by the platform under various formats (e.g. vtk, gmsh, med) or directly generated from a few input data. For instance, 2-D or 3-D Cartesian meshes as well as 2-D or 3-D honeycomb meshes can be created by just specifying the sizes of the domain and the number of cells.

For Cartesian meshes, Arcane provides a dedicated API giving access to neighbour elements in each direction. Cartesian meshes also come along with a dedicated block-refined AMR procedure.

Polyhedral meshes can currently be defined through the vtk format. This enables, for instance, the use of Voronoi meshes, see Figure 4. To handle such meshes, Arcane relies on the internal library Neo. This library is designed to handle complex polyhedral meshes that can possibly

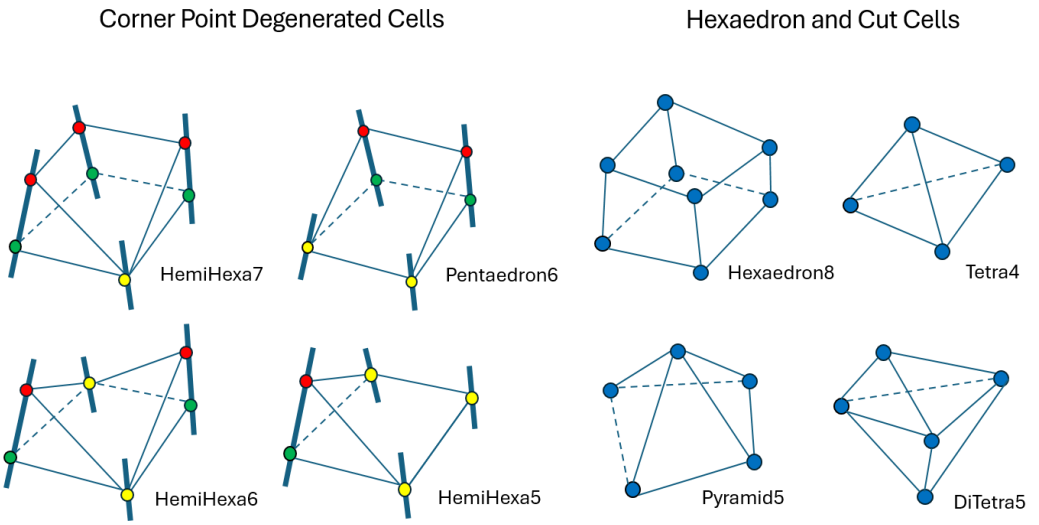


Figure 2. Degenerate hexahedral cells occurring in geological meshes.

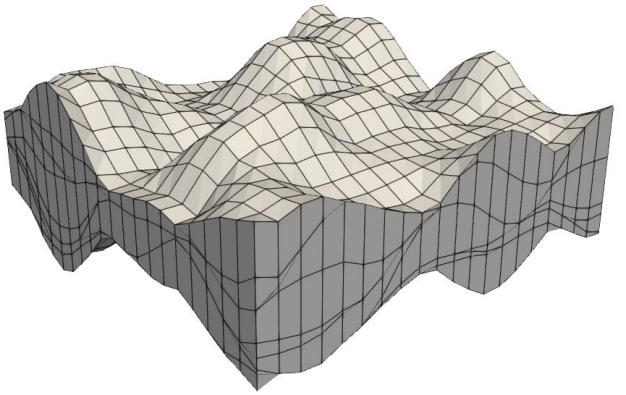


Figure 3. An example of geological mesh.

evolve during the simulation.[1] One of its interesting features is the way it asynchronously handles the mesh evolution by means of a dependency graph.

2.5. *A recent API for computing on accelerators*

Recently, a new API has been introduced in Arcane to handle accelerators with the following characteristics:

(i) it supports Cuda, HIP and SYCL backends;
(ii) it allows to use dynamically different types of GPU and multi-threaded CPU;
(iii) it ensures that the mesh connectivities and variables are available on the accelerator.

[1] Evolving meshes can, for instance, occur in geosciences when simulating the evolution of a sedimentary basin during geological times.

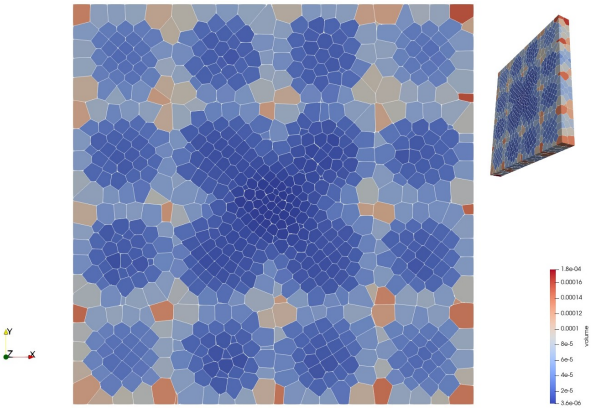


Figure 4. An example of Voronoi mesh colored by cell volumes.

```
// My mesh cell variables
Arcane::VariableCellReal m_a;
Arcane::VariableCellReal m_b;
Arcane::VariableCellReal m_c;

void compute_cpu_only()
{
 // given by Arcane
 Arcane::IMesh* mesh;









 // classical enumerate cell
 ENUMERATE_
    (Cell,icell,mesh->allCells()) {
  Real a = m_a(icell);
  Real b = m_b(icell);
  m_c(icell) = math::sqrt(a*a+b*b);
 };
}
```

Listing 3. Cell loop classical CPU version.

```
// My mesh cell variables
Arcane::VariableCellReal m_a;
Arcane::VariableCellReal m_b;
Arcane::VariableCellReal m_c;

void compute_cpu_or_gpu()
{
 // given by Arcane
 Arcane::IMesh* mesh;

 // define runner and queue
 // executing resource
 Arcane::Accelerator::Runner runner;
 // execution flow
 auto queue = makeQueue(runner);
 // computing kernel
 auto command = makeCommand(queue);

 // take for the device algorithm
 // 'a' and 'b' as inputs
 auto in_a = viewIn(command,m_a);
 auto in_b = viewIn(command,m_b);
 // 'c' as output
 auto out_c = viewOut(command,m_c);

 // Launch compute on device
 command << RUNCOMMAND_ENUMERATE
    (Cell,icell,mesh->allCells()) {
  Real a = in_a(icell);
  Real b = in_n(icell)
  out_c(icell) = math::sqrt(a*a+b*b);
 };
}
```

Listing 4. Cell loop accelerator version.

The API was inspired by the SYCL API from Khronos group [8]. As SYCL and Kokkos [9], it handles different architectures (NVIDIA, AMD, Intel). Its runtime-based approach allows for instance to address CPU (sequential or using multi-threading) or GPU or both with a same executable. The Arcane Accelerator API aims at providing the basic concepts from Arcane on GPU. The API allows to write classical mesh-based loops on GPU with few differences from the CPU version. In Listings 3 and 4 we compare a classical mesh compute loop on CPU and its implementation with accelerator API. The differences between both codes come only from a

dedicated setup needed on GPU, but the core of the loop itself is identical, see lines 28–31. In the GPU version, it is necessary to define a runner to specify the resource used for execution (line 13), a queue on this runner to manage the execution flow (line 15) and a computing kernel as a command (line 17). To manage variable access on specific accelerator devices, this API uses specific views of variables with access rules: In, Out or Inout, see lines 21–24.

This API manages Arcane variables on items (Cell, Node...), and gives access to the mesh connectivity. It also handles specific multi-dimensional array of dimension one to four. To submit command and process loop on target device we use specific loop macro based on C++ 11 lambda functions and specific loop macro, see Listing 5.

```
1  // pseudo-code describing the structure of the RUNCOMMAND_ API
2  command<<RUNCOMMAND_"LoopType"(args lambda consistent with "LoopType")
3   {
4     /* body lambda code to process on target device */
5   };
6
7  // two examples of the RUNCOMMAND_ API
8
9  // loop over mesh items (Cells, Nodes...).
10 // example for Nodes:
11 RUNCOMMAND_ENUMERATE(Node,inode,mesh()->allNodes()){};
12
13 // loop over multi-dimensional array of size one to four.
14 // example for one dimension:
15 RUNCOMMAND_LOOP1(iter, nb_value){};
```

Listing 5. Arcane API to execute parallel loops on a target device.

Finally, it also provides some high level functionalities like reductions and algorithms such as filtering, partitioning or sorting. It can also automatically profile the accelerator loops.

2.6. *From Arcane to a simulation application*

To build a numerical application upon the Arcane framework, the application developer will create *Modules* to implement the different physical models simulated. The entry-points of a *Module* structure the various computing steps of the model and are dynamically plugged into the application time-loop through a scheduler defined, by means of an xml `.config` file, by the application end-user.

Numerical and cross-functional tools are usually implemented in Arcane *Services*, so that they can be shared between *Modules* or even applications. These *Services* may handle numerical schemes, mesh reader/writer, etc. The library Alien, for instance, offers a collection of Arcane *Services* plugging different solver libraries, gathered under a unique interface.

Examples showing how to build an application upon Arcane are available in the platform repository [6] (tutorial or samples directories) or in mini applications arcane-benchs repository [10]. Two proxy-apps are also available in the Arcane Framework page on GitHub:

(i) the finite element proxy-app ArcaneFEM [11];
(ii) the porous media flow proxy-app ShArc [4], cf. Sections 4 and 5.

3. The ArcNum framework numerical tools

In this section, we present the ArcNum framework itself: a set of software components provided to facilitate the work of application developers. These components are located at the interface

between Arcane's kernel and the simulation application. These tools are for example plugged into the proxy-app ShArc [4], used in Sections 4 and 5 of this article.

The components of the framework ArcNum, namely *GUMP*, *Law Framework* and *AuDi*, are respectively dedicated to physical model handling, to the registration and evaluation of physical laws, and to automatic differentiation. They provide the needed tools to build the numerical core of an application in a highly readable and long-term maintainable way.

3.1. *GUMP data model*

For a given physical problem, GUMP (Generated Unified Manager Properties) allows to define the corresponding data model in an xml-style format file. This xml model has a very simple grammar defined in the `gump.xsd` scheme file. This grammar makes it possible to define a hierarchical structure of entities as shown in Listing 6 and, thus, to define all the physical properties carried by each entity of the model as in Listing 7. When the program is compiled, this file is parsed to generate C++ code that can be used by the application's developer. Even if most of the properties are defined in this xml model file, it is still possible to add additional properties within the application code.

```
<gump>
  <model namespace="ArcRes">
    <entity name="System">
      <contains>
        <entity name="FluidSubSystem" unique="true" />
      </contains>
    </entity>
    <entity name="FluidSubSystem">
      <contains>
        <entity name="FluidPhase" />
      </contains>
    </entity>
    <entity name="FluidPhase">
      <contains>
        <entity name="Species" />
      </contains>
    </entity>
    <entity name="Species">
      <contains>
        <entity name="Component" />
      </contains>
    </entity>
  </model>
</gump>
```

Listing 6. Example of an instance of a GUMP tree of entities.

```
 <entity name="FluidPhase">
  <supports>
    <property name="Viscosity" dim="scalar" type="real" />
    <property name="Density" dim="scalar" type="real" />
    <property name="CapillaryPressure" dim="scalar" type="real" />
  </supports>
</entity>
```

Listing 7. Example of GUMP properties associated to an entity.

This model is instantiated at runtime. In general, this action is carried out using a dedicated Arcane *Service*, see Section 2.2 and [1] for the description of Arcane *Services*. This *Service* allows the user to specify its model in the simulation input file, as shown in Listing 8.

```
<physical-model>
  <system name="UserSystem">
    <name>System</name>
    <fluid-system>
        <name>Fluid</name>
        <fluid-phase>
            <name>Water</name>
            <species>H2O</species>
        </fluid-phase>
        <fluid-phase>
          <name>Gas</name>
          <species>CO2</species>
        </fluid-phase>
    </fluid-system>
  </system>
</physical-model>
```

Listing 8. Example of an input simulation file instantiating a GUMP model.

This approach aims at bringing flexibility to flow models, for instance. Each property of the system is referred by a specific name called "xpath". It consists in specifying the property name with the path to the entity carrying it as a prefix. As an example, the input data of an initial condition *Service* is printed in Listing 9. The pressure property is uniquely designed through the path `[System]System::Pressure` and, here, its value is set to `1.38E+7`Pa.

```
<initial-condition name="Constant">
  <property>[System]System::Pressure</property>
  <condition>
    <value>1.38E+7</value>
  </condition>
</initial-condition>
```

Listing 9. Example of an input simulation file where xpath properties are used for initial conditions.

The generated C++ API also allows the application's developer to iterate over the model entities and manipulate the associated properties. An example of user code, parsing the physical model defined in Listing 9, is given in Listing 10. This code snippet prints the different entities of the physical model and shows how to access to the system pressure property by using the xpath we mentioned earlier.

```
ENUMERATE_SUBSYSTEM(isubsystem, system.subSystems()) {
    info() << *isubsystem; // prints: Fluid
    ENUMERATE_PHASE(iphase, isubsystem->phases()) {
      info() << *iphase;  // prints: Water Gas
      ENUMERATE_SPECIES(ispecies, iphase->species()) {
        info() << *ispecies; // prints: H20 CO2
      }
    }
}
// Access System Pressure
auto system_pressure = ArcRes::XPath::property(system,"System::Pressure");
```

Listing 10. C++ API to parse the physical system and access to the properties.

By defining a path to access to the different physical entities, GUMP provides a way to define the input and output properties of the law framework, described in the following section.

3.2. *Law framework*

When solving systems of partial differential equations, we frequently have to evaluate physical laws in groups of mesh items (cells, nodes...). For instance, in the context of porous-media flow simulations, these laws may correspond to petrophysical empirical relationships such as the relative permeabilities or the capillary pressures. The proposed law framework allows users to easily define new laws for their application.

In our framework, a law is simply a function which calculates output properties from input ones defined on groups of mesh items. It also evaluates the derivatives of the output properties with respect to the input ones.

In the law API, properties can be scalar, multi-scalar or vector-based. The access to a scalar law is simply done using its property name. It is for instance the case of the porosity. A multi-scalar law can be retrieved using several indexes of properties. Thus, the access to the molar fraction of a component is done using the name of the component and the name of the fluid phase.

The definition of a new law is carried out in three steps. First, the user defines the input and output properties of the law in an xml-style file (see Listing 11). This xml file is parsed at compilation time to generate a C++ header in the build directory. This header contains two classes, the `lawName::signature` defined in the xml file and the `lawName::law` class managing the evaluations of the law on different supports: mesh, mesh groups, item vectors or scalars. Second, the user has to implement the function which will be used to evaluate the law on a single mesh item (see Listing 12).

```
<law name="FluidDensityLawType">
  <input name="temperature" type="real" dim="scalar" />
  <output name="fluidDensity" type="real" dim="scalar" />
</law>
```

Listing 11. Example of xml file defining the law signature.

```
void FluidDensityLaw::eval(const Real T, Real& rho, Real& drho_dt)const
{
    rho = T + exp(T);
    drho_dt = 1 + exp(T);
}
```

Listing 12. User evaluation method corresponding to the xml definition.

At last, to be able to use a new law in an input `.arc` file, an arcane *Service* has to be created. This *Service* allows the user to choose the input and output properties at runtime. Listing 13 shows an example of `.arc` input file and Listing 14 is the implementation of the corresponding service. In this implementation, let us note that the two classes, generated in the header file, are instantiated: the `lawName::law` object taking in arguments the law evaluation function `FluidDensityLaw::eval` and the `law::signature` instance. At this stage the law is configured and added to the `function_mng` object (see last line of Listing 14). This last `Law::FunctionManager` object is used as a "law-database". When the application is initialized, each *Service*, which instantiates a law, adds it to the `Law::FunctionManager` object. The API also offers a `Law::FunctionEvaluator` object which allows one to select a subset of the available laws. This object is used to evaluate the selected laws on a specific support, at each time step,

for instance. Let us note that it is only at this stage that the evaluation support is specified. An example of law evaluation is given in Listing 15.

```
   <law name="FluidDensityLawConfig">
      <output>
        <fluid-density>[Phase]Water::Density</fluid_density>
      </output>
      <input>
        <temperature>[System]System::Pressure</temperature>
      </input>
      <parameters>
        <initial-temperature>0</initial-temperature>
      </parameters>
    </law>
```

Listing 13. Example of input simulation file configuring the law input and output.

```
#include "FluidDensityLawType_law.h" // generated from xml file
#include "FluidDensityLaw.h" // user definition of the law (cf. Listing 12)
void
FluidDensityLawConfigService::
configure(Law::FunctionManager& function_mng, ArcRes::System& system)
{
    // Initialize arguments from .arc input file
    auto fluid_density = options()->output().fluidDensity();
    auto temperature = options()->input().temperature();

    // Initialize parameters from .arc input file
    auto initial_temperature = options()->parameters().initialTemperature();

    // Set law arguments
    FluidDensityLawType::Signature signature;
    signature.fluidDensity = ArcRes::XPath::property(system, fluid_density);
    signature.temperature  = ArcRes::XPath::property(system, temperature);

    // Set law parameters
    FluidDensityLaw law;
    law.setParameters(initial_temperature);

    // Create law
    auto function = std::make_shared<FluidDensityLawType::Function>(
        signature, law, &FluidDensityLaw::eval);

    // Add law to function manager
    function_mng << function;
}
```

Listing 14. Example of C++ code to create a law and register it in the function manager.

```
// Prepare law evaluation
Law::FunctionManager function_mng; // Build and filled by the application
Law::FunctionEvaluator function_evaluator(function_mng);

IVariableMng variable_mng; // Given by the application
auto variable_accessor = variable_mng.variableAccessor<Cell>();

// Evaluate laws registered in function_mng
function_evaluator.evaluate(accessor, allCells(), Law::eWithDerivative);

// Access to law outputs
auto system_pressure = ArcRes::XPath::property(system,"System::Pressure");
auto computed_pressure = accessor.values(system_pressure);
auto computed_pressure_derivatives = accessor.derivatives(system_pressure);
```

Listing 15. Example of C++ code to evaluate the registered laws on a given support.

To conclude, the workflow to define and evaluate a physical law consists of four steps:

(i) definition of the law inputs and outputs in an xml-style file (Listing 11);
(ii) computation of its values and derivatives through a user-defined function (Listing 12);
(iii) definition of a configuration *Service* to instantiate and register the law object (Listing 14);
(iv) call to the law evaluator on a given support (Listing 15).

At runtime, users only have to complete the `.arc` file to set the actual inputs, outputs and parameters of the law, see Listing 13. After evaluating the laws, their values and derivatives can be read for any item included in the support defined in the evaluator. This functionality is a key point of the API since it provides all necessary data to build the matrices for the linear system assembly. These data are then gathered in *Contribution* objects which are described in the sequel.

3.3. *Contributions*

The resolution of nonlinear problems such as $\mathbf{F}(\mathbf{u}) = 0$ is often carried out thanks to a Newton's method. Each iteration k requires the construction of a linearized problem involving the Jacobian matrix of $\mathbf{F}$ and the residual vector $\mathbf{F}(\mathbf{u}^k)$. The *Contribution* API provides a convenient way to assemble the Jacobian and the residual terms, to compute automatically their derivatives to be stored in the matrix, and to add all these values in Alien systems.

3.3.1. *Contributions — Accessors*

For any GUMP property, the *Contribution* framework allows to recover its value and derivatives with respect to the main unknowns of the problem. When computing fluxes on a face, the derivatives often depend on various unknowns located around this face. This set of local elements is called a stencil. In the case of a diffusion flux discretized with the finite-volume two-point flux, this stencil includes the cells that share this face. To make the assembly of such fluxes easier in our framework, *Contributions* can be accessed according to the stencil of a given face. Let us give an example. In the Listing 16, lines 7–12 illustrate how to get *Contributions* related to three GUMP properties which are `ArcRes::Pressure`, `ArcRes::CapillaryPressure` and `ArcRes::FluidDensity`. These *Contributions* depend on an unknown manager that specifies the variables defined in each cell. This manager here appears twice since it is used with a two-point stencil. This stencil is initialized for each face on line 17. The following lines 18–25 show how to get the values for one stencil cell (back or front to the face).

```
1  _buildFluxInternal(ArcNum::Vector& residual, ArcNum::Matrix& jacobian)
2  {
3      // unknown manager of the system (List of unknown properties)
4      const auto& um = unknownsManager();
5
6      // contribution wrapper by property
7      auto P = Law::contribution<ArcRes::Pressure>(domain(),
8               functionMng(),um,um,system());
9      auto Pc = Law::contribution<ArcRes::CapillaryPressure>(domain(),
10              functionMng(),um,um,system().fluidSubSystem().phases());
11     auto Rho = Law::contribution<ArcRes::FluidDensity>(domain(),
12              functionMng(),um,um,system().fluidSubSystem().phases());
13
14     Arcane::FaceGroup inner_faces = mesh()->allCells().innerFaceGroup();
15     // Arcane loop on faces
16     ENUMERATE_FACE(iface,inner_faces) {
17         ArcNum::TwoPointsStencil stencil(iface);
18         const Law::Cell& cell_k = stencil.back();
19         const Law::Cell& cell_l = stencil.front();
```

```
        // GUMP loop on fluid phases
        ENUMERATE_PHASE(iphase, fluid.phases()) {
            // Acces contribution by mesh item [cell_k] or [cell_l]
            const auto Pk = P[cell_k];
            const auto Pl = P[cell_l];
            const auto Pck = Pc[iphase][cell_k];
            // idem for Pcl, Rhok and Rhol
            ...
        }
    }
}
```

Listing 16. Example of C++ code to access to the *Contributions*.

3.3.2. *Contributions — Combination and automatic differentiation with AuDi*

New *Contributions* can be created as a result of operations, for operators such as `*`, `/`, `+`, `-`, between several existing *Contributions*. An illustration is given in Listing 17. Here, the *Contribution* `grad_kl` is computed from the *Contributions* defined in the Listing 16. These operations make use of an automatic differentiation library, implemented in our framework and called AuDi.

```
_buildFluxInternal(ArcNum::Vector& residual, ArcNum::Matrix& jacobian)
{
    ...
        ENUMERATE_PHASE(iphase, fluid.phases()) {
            ...
            const auto grad_kl = ((Rhok + Rhol) / 2) *
            (Pk + Pck - Pl - Pcl);
            ...
            const auto flux_kl = ...// computed from grad_kl
        }
    }
}
```

Listing 17. Example of operations on *Contributions* computing values and derivatives.

AuDi is a C++ library based on generic programming techniques. It uses expression templates [12,13] and operator overloading. Operator overloading is a classical C++ technique that enables to define and implement operators on any object. It is applied in our case to implement the operators `+,-,/,*` on contributions. On the other hand, expression templates prevent the runtime from creating unnecessary variables or temporary objects and thus offer good CPU and memory performance in critical calculation sections such as flux evaluations.

3.3.3. *Contributions — Alien's linear system assembly*

Finally, adding *Contributions* to the Jacobian matrix or to the residual, defined by means of Alien's matrix and vector (see Section 2.3), can simply be done by using the operators `+=` or `-=`. Listing 18 shows an example where a finite-volume flux contribution `flux_lk` is inserted into the lines related to the equation `iequation` of both stencil cells. Let us notice that the insertion of the derivatives into the corresponding matrix columns is here done in a very compact way thanks to the use of the `stencil` object.

```
_buildFluxInternal(ArcNum::Vector& residual, ArcNum::Matrix& jacobian)
{
    ...
        ENUMERATE_PHASE(iphase, fluid.phases()) {
```

```
            ...
            const Arcane::Integer iequation = iphase.index();
            if (cell_k.isOwn()) {
                residual[iequation][cell_k]          += flux_kl;
                jacobian[iequation][cell_k][stencil] += flux_kl;
            }
            if (cell_l.isOwn()) {
                residual[iequation][cell_l]          -= flux_kl;
                jacobian[iequation][cell_l][stencil] -= flux_kl;
            }
        }
    }
}
```

Listing 18. Example of C++ code to assemble a Jacobian matrix using the *Contributions*.

4. First examples with ShArc, an Arcane-based demonstrator

In this section, we present numerical illustrations derived from two porous-media models implemented in ShArc: a single-phase thermohaline convection model and an isothermal two-phase flow model. We first introduce the common notations and subsequently provide detailed descriptions of both models.

4.1. *Notations*

The physical quantities used in both examples are introduced in this section. In the sequel, we also indicate their units where M stands for mass, L for length, T for time and Θ for temperature. The numerical values are given in the SI system. Vectors in $\mathbb{R}^3$ are denoted with bold letters and tensors in $\mathbb{R}^3 \times \mathbb{R}^3$ with blackboard-bold style letters.

We denote by $\phi[-]$ the porosity of the porous medium Ω, $\mathbb{K}[L^2]$ its permeability tensor. For a fluid phase p, $P_p[ML^{-1}T^{-2}]$ denotes its pressure, $\mu_p[ML^{-1}T^{-1}]$ the viscosity and $\rho_p[ML^{-3}]$ the mass density. We denote by $\boldsymbol{v}_p$ Darcy's velocity which is defined by

$$\boldsymbol{v}_p(P_p) = -\frac{\mathbb{K}}{\mu_p}(\nabla P_p - \rho_p \boldsymbol{g}) \tag{1}$$

where $\boldsymbol{g} = -g\nabla z[LT^{-2}]$ denotes the gravity vector.

4.2. *Thermohaline convection*

For the first model, we consider the transport of salt by water through the porous medium Ω. This problem, known as thermohaline convection [14], can be modelled by the following partial differential equations:

- the water mass balance,
$$\phi\partial_t\rho_\mathrm{w} + \mathrm{div}(\rho_\mathrm{w}\boldsymbol{v}_\mathrm{w}) = 0; \tag{2}$$
- the salt mass balance,
$$\phi\partial_t C + \mathrm{div}(\boldsymbol{F}_C) = 0, \tag{3}$$
where $\boldsymbol{F}_C = C\boldsymbol{v}_\mathrm{w} - \phi\mathbb{D}\nabla C$, $\mathbb{D}[L^2T^{-1}]$ is the porous-medium molecular diffusivity, $C[ML^{-3}]$ the mass concentration;

- the heat balance,

$$\partial_t\big((\phi c_{\mathrm{w}}\rho_{\mathrm{w}} + (1-\phi)\rho_{\mathrm{s}}c_{\mathrm{s}})\theta\big) + \mathrm{div}(\boldsymbol{F}_\theta) = 0 \tag{4}$$

where $\theta[\Theta]$ is the temperature,

$$\boldsymbol{F}_\theta = \rho_{\mathrm{w}}\theta\,\boldsymbol{v}_{\mathrm{w}} - \big((1-\phi)\mathbb{L}_{\mathrm{s}} + \phi\mathbb{L}_{\mathrm{w}}\big)\nabla\theta \tag{5}$$

and, for $p \in \{\mathrm{w},\mathrm{s}\}$ (the index s standing for salt), c_p is the heat capacity $[L^2T^{-2}\Theta^{-1}]$ and $\mathbb{L}_{\mathrm{p}}[MLT^{-3}\Theta^{-1}]$ the heat conductivity.

The boundary of Ω is partitioned into $\partial\Omega = \Gamma_{\mathrm{top}} \cup \Gamma_{\mathrm{bottom}} \cup \Gamma_{\mathrm{vertical}}$ where Γ_{top} is the highest part of the border, Γ_{bottom} the lowest one and $\Gamma_{\mathrm{vertical}}$ the remaining one. We denote by $\boldsymbol{n}$ the unit normal vector outwards to Ω. On the boundaries, we fix the following conditions:

$$\boldsymbol{v}_w \cdot \boldsymbol{n} = 0 \qquad \text{on } \partial\Omega, \tag{6}$$

$$\chi = \chi_{\mathrm{p}} \qquad \text{on } \Gamma_p,\ \chi \in \{C,\theta\},\ p \in \{\mathrm{top},\mathrm{bottom}\}, \tag{7}$$

$$\boldsymbol{F}_\chi \cdot \boldsymbol{n} = 0 \qquad \text{on } \Gamma_{\mathrm{vertical}},\ \chi \in \{C,\theta\}. \tag{8}$$

Initial conditions are also defined by zero for C and θ.

4.2.1. *Numerical setting*

We consider a thermohaline convection problem inspired from a test case proposed in [14]. Here the 2D space domain $\Omega = (0,300)\times(0,300)$ represents a homogeneous and isotropic aquifer where

- $\theta_{\mathrm{bottom}} = 200$, $\theta_{\mathrm{top}} = 0$, $C_{\mathrm{bottom}} = 10$, $C_{\mathrm{top}} = 0$;
- $\phi = 0.1$, $\mathbb{K} = 1e-13$, $\mathbb{D} = 1e-13$;
- $\rho_{\mathrm{s}} = 2700$, $c_{\mathrm{s}} = 1180$, $\mathbb{L}_{\mathrm{s}} = 2$;
- $c_{\mathrm{w}} = 4200$, $\lambda_{\mathrm{w}} = 0.65$;
- ρ_{w} is given by

$$\rho_{\mathrm{w}} = \rho_{\mathrm{w},0}\big(1 - \beta_\theta(\theta-\theta_0) + \beta_C(C-C_0)\big),$$

with $\rho_{\mathrm{w},0} = 1000$, $\beta_\theta = 3.37e-4$, $\beta_c = 6.38e-4$, $\theta_0 = 0$, $C_0 = 0$, $\mu_{\mathrm{w}} = 1e-3$.

A cell-centered finite volume discretization [15] is applied on the system (1)–(8), with an implicit Euler time stepping, a two-point flux approximation (TPFA) for the diffusive terms and upwinding for the convective ones. We perform the simulation on a grid discretized with 100×100 uniform cells. The initial time step is $\tau^0 = 8.64\times10^5\,\mathrm{s}$. At each time step we apply Newton's method [16] to solve the corresponding non-linear system. The linear system obtained at each Newton iteration is solved using an LU solver. If a divergence of the Newton algorithm occurs we divide the time step by 2. Otherwise, the time step is increased by multiplying it by 1.5. Figure 5 depicts the evolution of temperature and salt concentration at different time steps.

4.2.2. *Implementation within ShArc*

This section illustrates how the numerical tools presented in Section 3 are used to implement in ShArc the resolution of this thermohaline convection system.

Using GUMP (see Section 3.1), temperature and salt concentration are declared into the physical model by means of an xml file. An example of a section from this file is shown in Listing 19.

```
1    <property name="Temperature" dim="scalar" type="real" />
2    <property name="Concentration" dim="scalar" type="real" />
```

Listing 19. Definition of salt concentration and temperature GUMP properties.

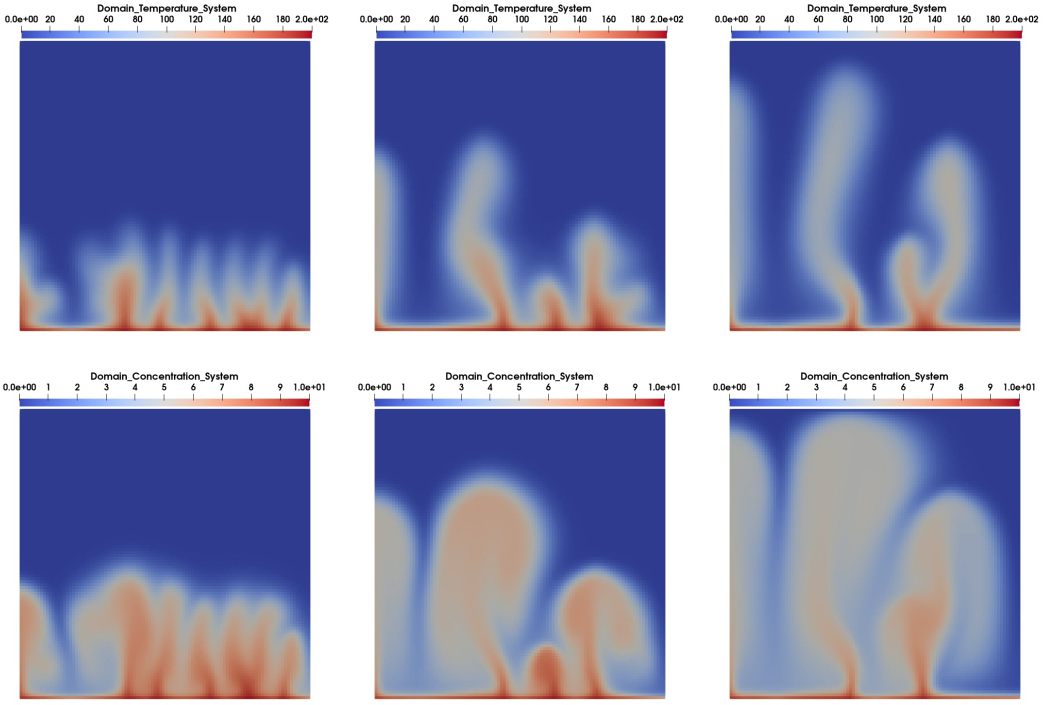


Figure 5. Evolution of the temperature (top) and salt concentration (bottom) after 28 years (left), 34 years (middle), and 40 years (right).

The law framework API, described in Section 3.2, allows to create the different laws related to the flow problem which is here considered. In our example, the fluid density ρ_w is defined by the code given in Listing 20. This piece of code will be evaluated in each mesh cell.

```
void FluidDensityLaw::eval(const Real T, const Real C, Real& rho,
                           Real& drho_dt, Real& drho_dc ) const
{
    Real factor = (1.0 - m_betat*(T - m_t0) + m_betac*(C - m_c0));
    rho = m_rho0 * factor;
    drho_dt = -1.0 * m_rho0 * m_betat;
    drho_dc = m_rho0 * m_betac;
}
```

Listing 20. C++ code computing fluid density law.

The parameters of the law ρ_w are then specified within the input file like in Listing 21. Note that the previous generated properties here appear as inputs of the law.

```
    <law name="FluidDensityLawConfig">
      <output>
        <fluid-density>[Phase]Water::FluidDensity</fluid-density>
      </output>
      <input>
        <concentration>[System]System::Concentration</concentration>
        <temperature>[System]System::Temperature</temperature>
      </input>
```

```
      <parameters>
        <rho0>1000</rho0>
        <betat>3.37e-4</betat>
        <betac>6.38e-4</betac>
        <t0>0</t0>
        <c0>0</c0>
      </parameters>
    </law>
```

Listing 21. Input simulation file defining fluid density law inputs, output and parameters.

When assembling the Jacobian matrix, contributions (see Section 3.3) provide the local values and derivatives of the variables and laws according to the primary unknowns of the system (here temperature and concentration) within the scheme stencil. Thus for a face σ between two cells K and L, the contributions contain the derivatives with respect to the unknowns of these two cells. Examples on how to get such contributions are given in Listing 22.

```
auto T = Law::contribution<ArcRes::Temperature>(domain(), ...,system());

auto phi = Law::values<ArcRes::VolumeFraction>(domain(),fluid);
auto mu = Law::contribution<ArcRes::Viscosity>(domain(), ...,fluid.phases());

auto rho = Law::contribution<ArcRes::Density>(domain(), ...,fluid.phases());
auto rhof = Law::contribution<FluidDensity>(domain(), ...,fluid.phases());

auto lambda_f = Law::contribution<HeatConductivity>( ..., fluid.phases());
auto lambda_s = Law::contribution<HeatConductivity>( ..., solid.phases());
```

Listing 22. C++ code retrieving the contributions for the evaluation of the thermal flux.

Finally, Listing 23 shows how to compute the discrete version of the thermal flux (5) and add it to the residual vector and to the Jacobian matrix.

```
const auto darcy_kl = (permeability / mu[fluid.phase(0)][cell_k])
        * tau[iface] * (P[cell_k] - P[cell_l] + rho_kl * dgz_kl);

const auto cond_s = ( (1.0-phi[K]) * L_s[solid.phase(0)][cell_k] +
                           phi[K]  * L_f[fluid.phase(0)][cell_k] );

const auto flux_t_kl =  darcy_kl
        * (one(darcy_kl >= 0)*T[cell_k]*rho[fluid.phase(0)][cell_k]
         + one(darcy_kl <  0)*T[cell_l]*rho[fluid.phase(0)][cell_l])
         + cond_s * tau[iface] * (T[cell_k] - T[cell_l]);

residual[iequation][cell_k]          += flux_t_kl;
jacobian[iequation][cell_k][stencil] += flux_t_kl;
```

Listing 23. C++ code building the upwind TPFA heat flux (5) using the contributions.

4.3. *Two-phase flow model*

We now consider an isothermal immiscible two-phase model of water and gas [17]. Each phase has only one component: H_2O and CO_2. Phase changes of these components are not here taken into account. Consequently, we use the indices "w" and "g" to refer indifferently to the corresponding phase or component. The difference between both phase pressures is equal to the capillary pressure:

$$P_g - P_w = P_{c_{g,w}}, \tag{9}$$

where $P_{c_{g,w}}[ML^{-1}T^{-2}]$ is a function of the gas saturation. Consequently, the model unknowns are the water pressure P_w and the saturations S_p, $p \in \{g, w\}$. Writing the conservation of the volume of each phase leads to the following system:

$$\phi \partial_t(\rho_w S_w) + \mathrm{div}(\rho_w k_{r,w} \boldsymbol{v}_w) = q_w, \tag{10}$$

$$\phi \partial_t(\rho_g S_g) + \mathrm{div}(\rho_g k_{r,g} \boldsymbol{v}_g) = q_g, \tag{11}$$

$$S_w + S_g = 1, \tag{12}$$

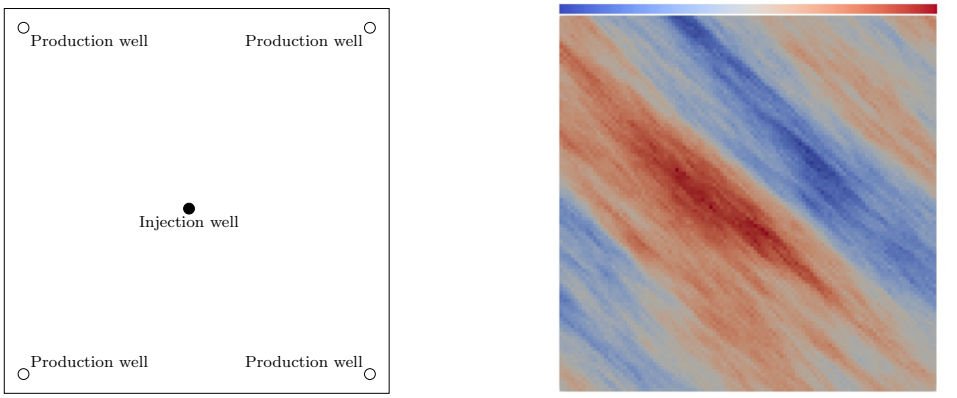
where $k_{r,p}(S_p)[-]$ stands for the relative permeability and $q_p[ML^{-3}T^{-1}]$ are sink or source terms which are non-zero where well perforations are present. No-flow boundary conditions are used for both phases. Initial conditions are given for S_g.

For further details about the two-phase flow model and the more general multiphase compositional flows in porous media we refer to [18–21]. More numerical experiments on practical problems illustrated on other Arcane-based IFPEN simulators with advanced resolution methods (adaptive mesh refinement, adaptive solvers, multilevel refinement, mixed multiscale finite elements) on general meshes can be found in [22–29].

4.3.1. *Numerical setting*

We consider the injection of CO_2 into a porous rock initially saturated with water. The spatial domain $\Omega = (0, 1010) \times (0, 1010)$ is discretized with a grid composed of 101×101 uniform cells. The process is organized following a five-spot pattern, see Figure 6(a), where one injection well is located in the middle of the domain and producers are in the four corners (these producers are here only used to create gradients of pressure through the domain for the purpose of our test). Peaceman well index models [17] are used to compute the discrete values of q_w and q_g where these wells are perforated (well radius is set to 0.5 and the skin factor is zero). This injection-production process is simulated during $t_F = 10$ years with an initial time step $\tau^0 = 8.64 \times 10^3$ s, which is equal to 0.1 days. If one time step iteration is completed, the time step is increased by a factor of 1.5. We divide it by 2 if a divergence of the Newton algorithm occurs.

The injection pressure is set constant in time and equal to $P_{inj} = 5.6e7$. In the same way, the producer pressure is fixed to $P_{pro} = 2.7e7$.



(a) Five-spot pattern **(b)** Permeability

Figure 6. Test-case configuration of Section 4.3.

The problem parameters are chosen as follows:

- $\phi = 0.1$, $\mathbb{K}$ is given by a geostatistical realization shown in Figure 6(b);
- $\rho_{\mathrm{w}} = 1025$, $\mu_{\mathrm{w}} = 1e-3$;
- $\rho_{\mathrm{g}} = 1.89$, $\mu_{\mathrm{g}} = 1.42e-5$;
- a Brooks–Corey model [30] was used for the petrophysical properties:
  - the relative permeability of a phase $p \in \{\mathrm{w}, \mathrm{g}\}$ is given by

$$k_{\mathrm{r},p}(S_p) = \begin{cases} 1 & \text{if } S_p \geq 1, \\ \left(\frac{S_p - S_p^{\mathrm{res}}}{1 - S_p^{\mathrm{res}}}\right)^n & \text{if } S_p^{\mathrm{res}} < S_p < 1, \\ 0 & \text{if } S_p \leq S_p^{\mathrm{res}}, \end{cases} \tag{13}$$

    where the residual saturations are respectively given by $S_{\mathrm{w}}^{\mathrm{res}} = 0.2$ and $S_{\mathrm{g}}^{\mathrm{res}} = 0.1$ and $n = 1$;
  - $P_{c_{\mathrm{g,w}}}$ is given by

$$P_{c_{\mathrm{g,w}}}(S_{\mathrm{g}}) = P_e \cdot (S_e)^m, \qquad S_e = 1 - \frac{S_{\mathrm{g}} - S_{\mathrm{g}}^{\mathrm{res}}}{1 - S_{\mathrm{w}}^{\mathrm{res}} - S_{\mathrm{g}}^{\mathrm{res}}},$$

    with $P_e = 8.73e5$, $m = -\frac{1}{2.89}$.

As for the previous test case, a two-point finite volume scheme with upwinding in space and the backward Euler scheme in time are applied to discretize the system (1), (10)–(12). Newton's method [16] along with LU solver is used to solve the resulting nonlinear system.

Figure 7 shows the evolution of the water saturation at three different time steps and, in particular, the faster front displacement towards the upper left and lower right corners where larger permeability values are present, see Figure 6(b) too.

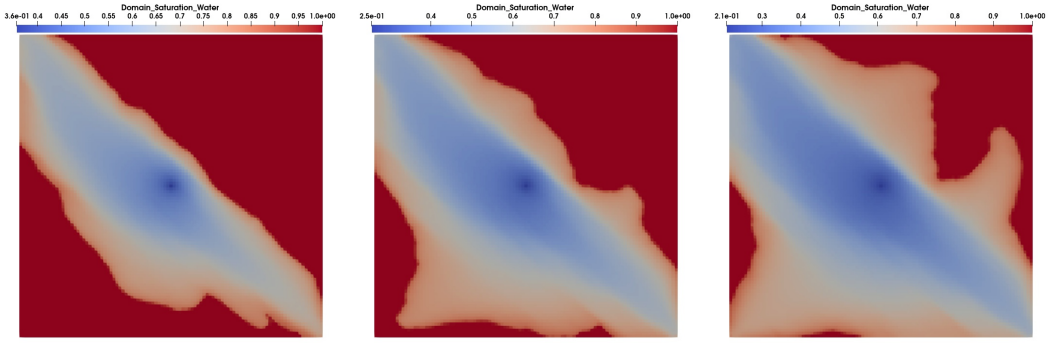


Figure 7. Water saturation at 0.5 year (left), 1 year (middle), and 1.5 year (right).

4.3.2. *Implementation within ShArc*

As for the previous example, this section gives an illustration of the use of the GUMP data model and of the law framework to define the capillary pressure: Listing 24 shows the line defining this property in GUMP, Listing 25 the C++ code defining the law, and Listing 26 the definition of the parameters of this law in the input file. At last, Listings 27 and 28 detail the construction of the *Contributions* required for the phase fluxes, their evaluation and their storage into the matrix and into the right-hand side of the linear system.

```
<property name="CapillaryPressure" dim="scalar" type="real" />
```

Listing 24. Definition of capillary pressure GUMP property.

```
void CapillaryPressureLaw::eval(const Real S, Real& Pc, Real& dPc_dS)const
{
    Real Se = Arcane::math::min(1 - (S - m_Sr)/(1 - m_Sr_ref -m_Sr), 1.0);
    Real dSe_dS = - 1.0/(1 - m_Sr_ref -m_Sr);
    Real alpha = - 1./m_lambda;
    Pc = m_Pe * pow(Se, alpha);
    dPc_dS = alpha * m_Pe * dSe_dS * pow(Se, alpha - 1);
}
```

Listing 25. C++ code computing the capillary pressure law

```
    <law name="CapillaryPressureLawConfig">
      <output>
        <capillary-pressure>[Phase]Gas::CapillaryPressure</capillary-pressure>
      </output>
      <input>
        <saturation>[Phase]Gas::Saturation</saturation>
      </input>
      <parameters>
        <Pe>8.73e5</Pe>
        <Sr-ref>0.2</Sr-ref>
        <Sr>0.1</Sr>
        <lambda>2.89</lambda>
      </parameters>
    </law>
```

Listing 26. Input simulation file defining the capillary pressure law parameters.

```
const Arcane::VariableFaceReal& T = m_transmissivities["T"];
auto P  = Law::contribution<ArcRes::Pressure>(...,system());
auto Pc = Law::contribution<ArcRes::CapillaryPressure>(...,fluid.phases());

auto mu  = Law::contribution<ArcRes::Viscosity>(...,fluid.phases());
auto rho = Law::contribution<ArcRes::Density>(...,fluid.phases());
auto kr  = Law::contribution<ArcRes::RelativePermeability>(...,fluid.phases());
```

Listing 27. C++ code retrieving the *Contributions* for the evaluation of the phase fluxes.

```
const auto grad_kl = T[iface] * (P[cell_k] + Pc[iphase][cell_k]
                                 - P[cell_l] - Pc[iphase][cell_l]);

const auto mobility_k =
        rho[iphase][cell_k] * kr[iphase][cell_k] / mu[iphase][cell_k];

const auto mobility_l =
        rho[iphase][cell_l] * kr[iphase][cell_l] / mu[iphase][cell_l];

const auto flux_kl = (audi::value(grad_kl)>=0) ? mobility_k*grad_kl
                                               : mobility_l*grad_kl ;

residual[iequation][cell_k]           += flux_kl;
jacobian[iequation][cell_k][stencil] += flux_kl;
```

Listing 28. C++ code building an upwind TPFA phase flux using the *Contributions*.

5. Massively parallel simulations on a simplified two-phase flow model

5.1. *Tests on* Irene *supercomputer*

We now present the first scalability tests performed on the Rome partition of *Irene* supercomputer [31]. The Rome partition contains 2292 dual-processor AMD Rome (Epyc) nodes at 2.6 GHz with 64 cores per processor, for a total of 293,376 computing cores. All nodes are connected through a HDR-100 Infiniband network.

The physical model used for these tests is similar to the one presented in Section 4.3. As a first attempt, it has been deliberately simplified in order to first test the code on larger grids than the ones usually used with other Arcane applications. This simplified model differs in terms of boundary conditions, rock and fluid properties. Namely, here:

- $\Omega = (0, L) \times (0, L) \times (0, H)$ with $L = 80000$, $H = 1000$, $t_F = 100$ days;
- the relative permeability of a phase $p \in \{w, g\}$ is given by (13) with $n = 2$ and $S_w^{res} = S_g^{res} = 0$, $P_{c_{g,w}} = 0$;
- $\phi = 0.2$, $\mathbb{K} = e^{f(x,y,z)}$ with

$$f(x, y, z) = -30 + 3 * \sin(2 * \pi * x/L) * \sin(2 * \pi * y/L) + y/L;$$

- $\rho_w = \rho_g = 1000$, $\mu_w = \mu_g = 10^{-3}$;
- $q_w = q_g = 0$, no-flow condition are used on $\partial\Omega$ except on $x = 0$ and $x = L$ where $P\big|_{x=0} = 8.10^7$, $S_g\big|_{x=0} = 1$, $P\big|_{x=L} = 10^7$.

The computation domain Ω is discretized with a Cartesian grid of size $N^2 \times N_z$ with $N = 2048$ and $N_z = 100$. The time step evolution was chosen according to the evolution of Newton's iterations starting with an initial time step of one day with a maximal value equal to 10 days. The linear systems were solved with a BiCGStab method preconditioned with CprAMG (Hypre Boomer AMG + block Jacobi ILU0). Note that the defined grid and model lead to ≈ 1.258 billion equations at each Newton iteration. Starting from 10 nodes, MPI parallel simulations were carried out up to 128 nodes (16,384 cores) without changing the grid size (strong scalability study). Figure 8 shows the scalability of ShArc's time loop from 10 up to 128 nodes (blue curve) and the theoretical linear performance compared to the 10-node run (red curve) which is the lowest number of nodes that can run this test case. These results were obtained as part of the Genci *Grands Challenges 2024* [32]. We observe a super-linear scalability of ShArc performances up to 100 nodes due to better cache usage as per core problem size decreases. On this range of nodes, this study has highlighted a parallel efficiency of ShArc's time loop at around 90%.

Secondly, simulations have also been carried out at the limit of the Alien platform's capacity:[2] a grid containing 700 million of elements or so was distributed on 256 nodes (32,768 cores and 2.1 billion equations on the whole). These tests performed on Irene supercomputer have also provided an opportunity to compare the performances of Alien's MCGSolver in both MPI and MPI-OpenMP contexts. Further details are given in [33]. The input data file of these case can be found on GitHub [34].

5.2. *Tests on IFPEN* Ener440 *supercomputer*

In this second series of parallel tests, we take the model a step further introducing a capillary pressure P_c, given in the description below. The tests are executed on IFPEN *Ener440* supercomputer, composed of 240 computing nodes with dual Intel Skylake-X G-6140 processors clocked at 2.3 GHz. Each node holds 2×18 computing cores so this supercomputer provides a total of 8640 computing cores. All nodes are connected with an Intel Omni-Path high speed network.

[2] Due to the equation indexing with 32-bit integers.

Figure 8. Scalability of ShArc's time loop from 10 up to 128 nodes (blue curve) and the theoretical linear performance compared to the 10-node run (red curve).

The physical model used for these tests is close to the model used with *Irene* supercomputer, but with a capillary pressure P_c. It has the following characteristics:

- $\Omega = (0,L)\times(0,L)\times(0,H)$ with $L = 20000$, $H = 100$, $t_\mathrm{F} = 1000$ days;
- $\rho_\mathrm{w} = \rho_\mathrm{g} = 1000$, $\mu_\mathrm{w} = 10^{-3}, \mu_\mathrm{g} = 10^{-4}$;
- $q_\mathrm{w} = q_\mathrm{g} = 0$, no-flow condition are used on $\partial\Omega$ except on $x = 0$ and $x = L$ where $P\big|_{x=0} = 6.10^7$, $S_\mathrm{g}\big|_{x=0} = 1$, $P\big|_{x=L} = 10^7$;
- $P_{c_\mathrm{g,w}}$ is given by

$$P_{c_\mathrm{g,w}}(S_\mathrm{g}) = P_e\cdot(S_e)^m, \qquad S_e = 1 - \frac{S_\mathrm{g} - S_\mathrm{g}^\mathrm{res}}{1 - S_\mathrm{w}^\mathrm{res} - S_\mathrm{g}^\mathrm{res}},$$

with $P_e = 10^5$, $m = -1$, $S_\mathrm{w}^\mathrm{res} = 0.2$ and $S_\mathrm{g}^\mathrm{res} = 0.1$.

The domain Ω is discretized with a Cartesian grid of size $N^2 \times N_z$ with $N = 2048$ and $N_z = 10$.

The performance results in Figure 9 are similar to those observed with *Irene* system but at a lower scale: N_z is reduced from 100 to 10 and tests are performed up to 20 nodes (720 cores). Relative efficiency, compared to the 3 nodes test, is > 94%. The test case is also available at [34].

6. Conclusion

The ArcNum framework presented in this article offers easy-to-use numerical tools significantly simplifying the writing of numerical methods within scientific computing codes. These numerical tools are built upon the Arcane platform, providing mesh management, data structures and parallelism. For linear system resolutions, they rely on the Alien library, giving access to a wide range of linear solvers. Used together, ArcNum, Arcane, and Alien offer a complete toolbox for building numerical simulation applications. This framework has been used to build the open

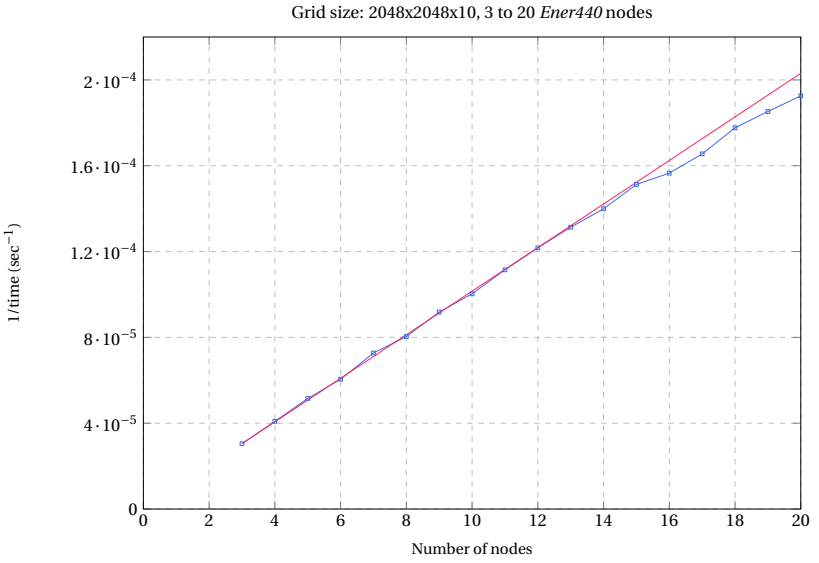


Figure 9. Scalability of ShArc's time loop from 3 up to 20 nodes (blue curve) and the theoretical linear performance compared to the 3-node run (red curve).

source proxy-app ShArc for Geosciences simulation. The two case studies presented in this publication and conducted with ShArc highlight the flexibility, development time savings and robustness of the ArcNum framework. A scalability study shows that this gain is not at the expense of parallel performance.

The ArcNum framework is also used within complex industrial applications at IFPEN. For example, it is the foundation of a compositional multiphase reactive transport simulator, coupled with mechanics. In this context, more advanced schemes, such as (non)linear multipoint finite volume and virtual element methods, have been implemented [35,36]. In this much more demanding context, the framework has been used with more complex data models, a great number of physical laws and varied assembly algorithms. It was therefore necessary to add new functionalities to ensure the needed expressiveness and maintain the performance requirement. More precisely, the following features were added to the framework for internal use at IFPEN:

- the ability to create a graph of laws to enable the composition of physical laws;
- the management of dynamic multi-point stencils;
- a sparse automatic differentiation.

The next step for this framework will concern the physical law package, where the usage of AI trained models and the ability to compute asynchronously on accelerators will be added.

Declaration of interests

The authors do not work for, advise, own shares in, or receive funds from any organization that could benefit from this article, and have declared no affiliations other than their research organizations.

References

[1] G. Grospellier and B. Lelandais, "The Arcane Development Framework", in *POOSC '09: Proceedings of the 8th Workshop on Parallel/High-Performance Object-Oriented Scientific Computing* (K. Davis, ed.), ACM Press, 2009, (11 pages).

[2] CEA/IFPEN, *The Arcane Framework organization*. Online at https://github.com/arcaneframework (accessed on November 3, 2025).

[3] J.-M. Gratien, C. Chevalier, T. Guignon, X. Tunc, P. Have and S. de Chaisemartin, "Evaluation of the performance portability layer of different linear solver packages with ALIEN, an open generic and extensible linear algebra framework", Conference paper : The 8th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS Congress 2022), 2022. Online at https://www.scipedia.com/public/gratien_et_al_2022a.

[4] CEA/IFPEN, *The ShArc proxy-app repository*. Online at https://github.com/arcaneframework/sharc.git (accessed on November 3, 2025).

[5] Inria, *The ShArc proxy-app installation guide repository*. Online at https://gitlab.inria.fr/numpex-pc5/wp2-co-design/proxy-sharc.git (accessed on November 3, 2025).

[6] CEA/IFPEN, *The Arcane platform repository*. Online at https://github.com/arcaneframework/framework.git (accessed on November 3, 2025).

[7] A. Anciaux-Sedrakian, P. Gottschling, J.-M. Gratien and T. Guignon, "Survey on Efficient Linear Solvers for Porous Media Flow Models on Recent Hardware Architectures", *Oil Gas Sci. Technol.* **69** (2014), pp. 753–766.

[8] The Khronos Group, *SYCL*, 2020 Specification. Online at https://www.khronos.org/sycl/ (accessed on November 3, 2025).

[9] H. C. Edwards, C. R. Trott and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns", *J. Parallel Distrib. Comput.* **74** (2014), no. 12, pp. 3202–3216.

[10] CEA/IFPEN, *Arcane benchs and mini apps repository*. Online at https://github.com/arcaneframework/arcane-benchs.git (accessed on November 3, 2025).

[11] CEA/IFPEN, *The ArcaneFEM proxy-app repository*. Online at https://github.com/arcaneframework/arcanefem.git (accessed on November 3, 2025).

[12] T. Veldhuizen, "Expression Templates", *C++ Rep.* **7** (1995), no. 5, pp. 26–31.

[13] D. Vandevoorde and N. Josuttis, *C++ Templates: The Complete Guide*, Addison-Wesley Publishing Group, 2002.

[14] H.-J. G. Diersch and O. Kolditz, "Coupled groundwater flow and transport: 2. Thermohaline and 3D convection systems", *Adv. Water Resources* **21** (1998), no. 5, pp. 401–425.

[15] R. Eymard, T. Gallouët and R. Herbin, "The finite volume method", in *Solution of Equation in $\mathbb{R}^n$ (Part 3), Techniques of Scientific Computing (Part 3)* (P. G. Ciarlet and J.-L. Lions, eds.), Handbook of Numerical Analysis, vol. 7, Elsevier, 2000, pp. 713–1020.

[16] C. T. Kelley, *Solving nonlinear equations with Newton's method*, Society for Industrial and Applied Mathematics, 2003.

[17] Z. Chen, *Reservoir simulation: mathematical techniques in oil recovery*, Society for Industrial and Applied Mathematics, 2007.

[18] G. Acs and E. Farkas, "General Purpose Compositional Model", *SPE J.* **25** (1985), no. 4, pp. 543–553.

[19] K. Aziz and A. Settari, *Petroleum Reservoir Simulation*, Applied Science Publishers, 1979.

[20] K. H. Coats, "An Equation of State Compositional Model", *SPE J.* **20** (1980), no. 5, pp. 363–376.

[21] L. C. Young and R. E. Stephenson, "A Generalized Compositional Approach for Reservoir Simulation", *SPE J.* **23** (1983), no. 5, pp. 727–742.

[22] A. Anciaux-Sedrakian, L. Grigori, Z. Jorti and S. Yousef, "Adaptive linear solution process for single-phase Darcy flow", *Oil Gas Sci. Technol.* **75** (2020), article no. 54 (11 pages).

[23] D. A. Di Pietro, E. Flauraud, M. Vohralík and S. Yousef, "A posteriori error estimates, stopping criteria, and adaptivity for multiphase compositional Darcy flows in porous media", *J. Comput. Phys.* **276** (2014), pp. 163–187.

[24] D. A. Di Pietro, M. Vohralík and S. Yousef, "An a posteriori-based, fully adaptive algorithm with adaptive stopping criteria and mesh refinement for thermal multiphase compositional flows in porous media", *Comput. Math. Appl.* **68** (2014), no. 12, Part B, pp. 2331–2347.

[25] J.-M. Gratien, O. Ricois and S. Yousef, "Reservoir Simulator Runtime Enhancement Based on a Posteriori Error Estimation Techniques", *Oil Gas Sci. Technol.* **71** (2016), no. 5, article no. 59 (11 pages).

[26] G. Mallik, M. Vohralík and S. Yousef, "Goal-oriented a posteriori error estimation for conforming and nonconforming approximations with inexact solvers", *J. Comput. Appl. Math.* **366** (2020), article no. 112367 (20 pages).

[27] M. Vohralík and S. Yousef, "A simple a posteriori estimate on general polytopal meshes with applications to complex porous media flows", *Comput. Methods Appl. Mech. Eng.* **331** (2018), pp. 728–760.

[28] S. Yousef, "A posteriori-based, local multilevel mesh refinement for the Darcy porous media flow problem", *J. Comput. Appl. Math.* **454** (2025), article no. 116162 (10 pages).

[29] M. A. Puscas, G. Enchéry and S. Desroziers, "Application of the mixed multiscale finite element method to parallel simulations of two-phase flows in porous media", *Oil Gas Sci. Technol.* **73** (2018), article no. 38 (14 pages).

[30] R. J. Brooks and A. T. Corey, *Hydraulic properties of porous media*, Hydrology Paper, Colorado State University, no. 3, 1964.

[31] CEA, "Supercomputer architecture", in *TGCC public documentation, version 2025-10-10.1612*, 2025. Online at https://www-hpc.cea.fr/tgcc-public/en/html/toc/fulldoc/supercomputer_architecture.html (accessed on November 3, 2025).

[32] GENCI, "Grands Challenges Scalaires Joliot-Curie, Adastra 2024", in *Grands Challenges*. Online at https://www.genci.fr/grands-challenges#ui-id-1 (accessed on November 3, 2025).

[33] A. Anciaux-Sedrakian, R. Gayno, T. Guignon and A. K. Mohamed El Maarouf, "Efficient high-fidelity simulations for the energy transition using ARM and x86 64 architectures", Conference paper: The 9th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS Congress 2024), 2024. Online at https://www.scipedia.com/public/Anciaux-Sedrakian_et_al_2024a.

[34] CEA/IFPEN, *Sharc: Two phase flow test cases*. Online at https://github.com/arcaneframework/sharc/tree/main/test/LargeScaleTwoPhaseFlowSimulation/ (accessed on November 3, 2025).

[35] M. Schneider, L. Agélas, G. Enchéry and B. Flemisch, "Convergence of nonlinear finite volume schemes for heterogeneous anisotropic diffusion on general meshes", *J. Comput. Phys.* **351** (2017), pp. 80–107.

[36] G. Enchéry and L. Agélas, "Coupling linear virtual element and non-linear finite volume methods for poroelasticity", *Comptes Rendus. Mécanique* **351** (2023), no. S1, pp. 395–410.

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus. Mécanique
2025, Vol. 353, p. 1315-1349
https://doi.org/10.5802/crmeca.337

Research article

Modular and Interdisciplinary Methods for Aeroelastic Simulations (MIMAS)

Antoine Riols-Fonclare [a], Yann Vallat [a], Pierre-Emmanuel Des Boscs [a], Antoine Placzek [a], Alain Dugeai [a], Cédric Liauzun [a], Christophe Blondeau [a], Charly Mollet [a] and Mikel Balmaseda [b]

[a] DAAA, ONERA, Institut Polytechnique de Paris, 92320 Châtillon, France

[b] DAAA, ONERA, Institut Polytechnique de Paris, 92190 Meudon, France

E-mail: antoine.riols-fonclare@onera.fr

Abstract. In the context of high aspect ratio wings or blades, aeroelasticity is becoming increasingly crucial for predicting the safety, efficiency, and performance of modern aircraft. This paper describes the development of MIMAS, a computational framework for simulating complex nonlinear aeroelastic phenomena from incompressible to highly transonic flows, in steady or unsteady configurations. On the one hand, MIMAS features advanced mesh deformation and transfer algorithms that have been renewed and optimised to enable faster computations compared to previous implementations within ONERA legacy codes. On the other hand, it offers ready-to-use coupling scenarios to support loose to strong fluid-structure interactions. This environment provides a high-level end-user abstraction layer that allows to couple a wide range of aerodynamic and non-linear structural simulation codes. In addition, the data structure is sufficiently generic to handle both structured and unstructured discretizations. In this paper, we first demonstrate the ability of MIMAS to reproduce existing representative computations such as harmonic forced motion, static coupling and dynamic coupling, without overhead induced by the modular implementation. Second, we present new capabilities of MIMAS, in particular the improved algorithms for mesh deformation and data transfer and its ability to leverage modern HPC architectures.

Keywords. Aeroelasticity, numerical simulations, coupling, mesh deformation.

Note. Article submitted by invitation.

Manuscript received 15 January 2025, revised 4 August 2025, accepted 24 October 2025.

1. Introduction

Designing efficient aircraft while minimizing environmental impact requires the ability to simulate highly complex unsteady physical phenomena. The latest trend, led by industry-leading companies such as Safran and Airbus, involves the use of high aspect ratio blades or wings often made of composite materials, for which non-linear aeroelastic phenomena need to be modelled accurately. Notable examples include the open rotor and stator engine developed by SAFRAN/GE Aviation in the framework of the RISE project and the X-WING plane developed by Airbus in the framework of the eXtra Performance Wing project. ONERA is also deeply involved in the hydrogen-powered aircraft (Gullhyver research project). These innovative architectures, with enhanced flexibility, must meet a wide range of structural and aeroelastic design constraints to ensure both safety and performances.

For example, in modern aircraft configurations, several technological components such as struts, morphing winglets or folding wing tips are planned to be employed to improve performance and support highly deformable wings. The interplay between all these components introduces a high level of complexity, which must be untangled. During critical flight phases (e.g., take-off and manoeuvres), unsteady aerodynamic forces engage with engine structures, generating periodic stresses that may shorten the lifespan of blade components. Accurate modelling of these forces aids in assessing fatigue-related risks and optimizing blade lifespan. Phenomena like whirl flutter are also crucial in designing the new generation of propellers. This form of aeroelastic instability can induce severe vibration amplitudes on the nacelle, potentially leading to structural failure if not adequately controlled [1–3].

Various approaches exist to address these challenges. Analytical methods and reduced models are commonly used due to the computational cost of high-fidelity coupled aeroelastic analyses. Although the rapid methods are efficient to explore parameter space in early-stage designs, they lack the accuracy to capture relevant physical phenomena. For detailed analyses, high-fidelity simulations are preferred, and there is a growing trend toward integrating multi-fidelity approaches for aircraft design and optimization [4]. In the era of exascale supercomputers, high-fidelity simulations raise the hope of predicting the entire aircraft or engine dynamics with all its detailed technological effects and inherent complexity. However, several factors currently limit these simulations, including code performance, interoperability, and limited physical modelling. To overcome these limitations, High-Performance Computing (HPC) codes designed to solve Fluid-Structure Interactions (FSI) using modern and optimized algorithms have become essential. In this context, ONERA has initiated the development of the SoNICS code (property of ONERA/Safran) for the fluid part [5] with the ambition of running this code on exascale machines with much better performance than existing codes (elsA, property of ONERA/Safran, [6]). The ONERA, DLR and Airbus collaboration is also deeply involved in the implementation of CODA [7], a new Computational Fluid Dynamics (CFD) code. Yet, coupling aerodynamical and structural problems needs additional key operations. These include transferring fields between both grids, deforming the fluid mesh during selected physical iterations, and ultimately coupling the solvers with optimised time-marching strategies. The latter involves in particular the development of new methodologies for solving non-linear coupled problems such as mixed end-point or mid-point temporal schemes, harmonic balance methods for periodic flows [8–11], asynchronous solvers [12], or ultimately the monolithic approach [13].

Historically, high-fidelity aeroelastic simulations at ONERA were performed directly within the elsA code, through a dedicated module [14]. All aeroelastic components are integrated directly within the code, following the approach of software like SU2 [15]. While elsA remains a powerful tool for such simulations, the increasing complexity of modern aerospace configurations is exposing its limitations. The elsA aeroelastic module is inherently restricted to structured grids and is not suited for analyzing non-linear structural behaviors. For static simulations, the structure's representation is via a condensed flexibility matrix while dynamic calculations rely on modal basis projections, which limits the ability to simulate nonlinear structural dynamics. Temporal mesh deformation is generally linearly interpolated due to the high computational cost of the existing techniques. Interpolation techniques for field transfers (e.g., forces, displacements) lack parallelization and are not optimized for large scale simulations. For time-marching, a basic fixed-point method is employed with a static relaxation parameter and coupling frequency, which may not be the most efficient way to couple fluid and structure. Finally, coupling efficiently with external Computational Structural Mechanics (CSM) codes such as NASTRAN is also challenging inside elsA, and adapting the code for that has become increasingly difficult.

To overcome these limitations and upgrade the aeroelastic coupling capabilities, subsequent developments have been performed to externalize and improve the different components outside from elsA's kernel. These components comprise mesh deformation techniques, transfer algorithms, mechanical solver and the driver that controls the time-marching iterations. This modular approach offers greater flexibility in terms of coupling and also makes it possible to work not only with elsA but also with other CFD codes, while reusing the same components for coupling. The modularization also provides a new way to tackle emerging challenges, such as simulating highly flexible structures in conjunction with flight dynamics, and their interaction with control systems and aeroelastic components, all of which are critical for modern aircraft. While simulations of this nature have already been conducted [16–18], they are often constrained by high computational costs. To enable more efficient calculations, it is clear that solvers must be externalized within a modular framework in which algorithms can be easily tested and improved.

Most of the external algorithms and tools have been then regrouped into a unified Python environment called MIMAS for *Modular and Interdisciplinary Methods for Aeroelastic Simulations*. The main goals of this library are to:

(1) take into account more complex physics, particularly structural non-linearities (e.g. contact, large displacements);
(2) improve the global performance of aeroelastic calculations and fit with the current HPC criteria;
(3) handle structured and unstructured mesh discretizations (the latter being crucial for simulating technological components);
(4) ensure the coupling and interchangeability of a large variety of CFD/CSM codes;
(5) extend to more sophisticated coupling (considering flight dynamics, trim, control);
(6) offer high flexibility in the developments of new methodology and the coupling between new components.

Note that the proposed approach is similar to existing coupling libraries such as preCICE [12] dedicated to partitioned multi-physics simulations, including, but not restricted to Fluid-Structure Interaction (FSI), and ParaSiF_CF [19] developed by University of Manchester and dedicated to the resolution of massively parallel partitioned FSI problem. The MDO Lab is also strongly involved in the development of a modular platform integrating aerodynamic (ADflow) and structural (TACS) solvers, and coupling these codes to perform shape optimization (OpenMDAO) [20].

The remainder of this article is structured as follows: in Section 2 a comprehensive overview of the MIMAS library and its alignment with the outlined objectives is given. In particular, the main evolution compared to previous elsA implementations is highlighted and the novel numerical methods employed within the library are detailed, focusing particularly on mesh deformation techniques and transfer algorithms. In Sections 3 and 4, a thorough validation and testing of the library across various industrial and academic configurations, encompassing airplanes, turbomachinery and helicopters, is presented.

2. Capabilities and numerical improvements beyond elsA

In its current version, MIMAS is capable of simulating most of the problems that have historically been addressed by elsA's aeroelastic module (AEL). But more importantly, it features new algorithms and optimized, parallelized versions of the existing ones. It operates with both structured and unstructured meshes (which was not the case with elsA), and has minimal dependence on external libraries, yet taking benefice from the current expertise and HPC libraries available at ONERA. MIMAS is today able to solve complex static or dynamic problems on industrial configurations, some including nonlinear structures in transonic flows. To achieve this, MIMAS has been

interfaced with the new generation of CFD codes developed at ONERA such as SoNICS or CODA and finite element codes like NASTRAN, TACS [21] or Code_Aster [22] enabling the achievement of high fidelity aeroelastic simulations. It incorporates advanced parallelized algorithms for mesh deformation and the transfer of displacements and forces, ensuring lower CPU cost at each coupling step. The integration within a PyTree[1] coupling environment also ensures that the entire framework can manage the intricate data structures and workflows required by HPC fluid solvers. This encompasses shared-memory coupling and swift data search capabilities. Finally, the modular and object-oriented approach allows users to assemble components like building blocks in a LEGO set, helping them customize physical problems to their specific needs. In the following sections, we detail each new capability of MIMAS and provide a brief overview of the numerical methods that are used.

2.1. *A modern and modular architecture*

MIMAS is based on a modular architecture for which the fluid (CFD) and structural (CSM) codes are both external and exchange data through a coupling tree. The coupling tree is a Python structure enforcing the CFD General Notation System (CGNS) and sharing the data in memory with both the CFD and CSM codes, whenever possible. These data comprise the fluid and structural geometries (mesh, normals, surfaces, volumes, etc.), and the physical fields of interest (local forces, torques, displacements, structural modes...). Memory-sharing capabilities allow data to be stored efficiently without duplication, enabling quick access for both HPC codes and the coupling library. Dedicated methods have been developed to write data into the tree or access rapidly the fields from the tree in the numpy format (or dictionary format) without additional memory cost. Another advantage of this structure is that it supports parallel processing, enhancing the efficiency of coupled simulations. The transfer and deformation modules of MIMAS, as well as the dedicated module for geometrical calculations, operate independently from the coupling tree but retrieve data through it. Note that to exchange data with software such as NASTRAN, which may be installed on a remote machine, we use the Python module Pyro, which enables remote data communication. For NASTRAN in particular, the coupling is performed in-memory via the OpenFSI module, provided as part of the MSC Software suite. Hopefully, the resulting complex infrastructure is hidden from the user, who interacts only with the high-level coupling tree interface.

Aeroelastic calculations are then built around a central Python driver that executes an iteration loop between external CFD and CSM solvers (depicted as orange boxes in Figure 1). A common interface has been developed for each fluid and structural code, which enables to compute a single iteration or time step, restart the resolution process at a desired iteration, extract relevant data for aeroelastic coupling and update geometry/meshes from the Python framework. We emphasize that with MIMAS, users have the freedom to create their own drivers by constructing dedicated objects (models, coupling trees, mesh deformers, interpolators and solvers). They can manipulate these objects at each step of the aeroelastic loop to customize their calculations. Obviously, this freedom of action would not be possible inside compiled codes like elsA.

2.2. *Dealing with unstructured meshes*

Unstructured grids are increasingly prevalent both in industry and in academia due to their flexibility and ability to accurately represent complex geometries with a reasonable number of cells. Nevertheless, structured grids remain used for specific simulations, so there is today a

[1] A tree-like structure in Python with data stored in the leafs (similar to CGNS format).

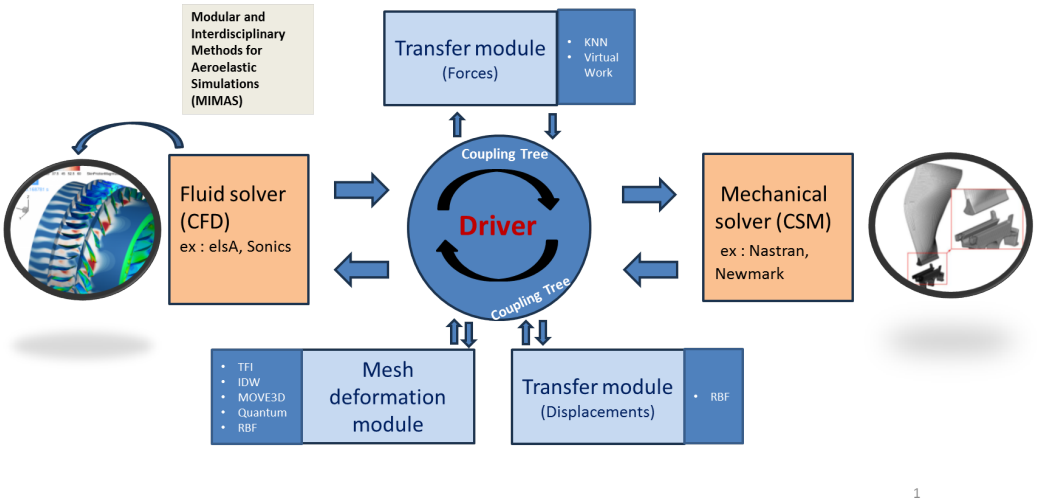


Figure 1. Modular architecture of MIMAS.

need for a coupling environment to handle both structured and unstructured meshes. To address these needs, we designed MIMAS from the very beginning to effectively handle both kind of grid representations.

Unstructured meshes in coupled problems can be challenging due to the multitude of formats available for representing mesh connectivity. We can for instance cite the polyhedral "NGon" format, used by several CFD codes (elsA, SoNICS) or the more classical "elements" format used by CODA and finite-element codes like NASTRAN. To deal with this multitude of formats, a dedicated mesh module has been implemented inside MIMAS to convert unstructured meshes into a common mesh object format. Geometrical quantities can therefore be computed regardless the initial mesh format. We implemented inside MIMAS an entire library in this unstructured framework able to compute geometrical quantities (such as volume, cells areas, face and cells centered quantities) and cell quality criteria (such as shear, aspect ratio, skewness...). Deformation, interpolation and transfer modules are further based on this internal library. At the moment, the library is able to deal with linear elements (TRI3, QUAD4, TETRA4, PYRA5, WEDGE6, HEXA8). In order to be fully generic an interface has been developed with the VTK (Visualization Toolkit) library which provides a wider range of geometric mesh calculation capabilities and many filtering options, including surface extraction. One of the key strengths of VTK is its ability to handle higher order elements. However, for the applications described in this paper, our internal library is largely sufficient. Unitary tests have been performed to check that deformation and transfer algorithms behave similarly in the structured or unstructured configurations (though, not shown in this paper).

2.3. *Improved mesh deformation algorithms*

A key step in aeroelastic calculations involves deforming the fluid mesh to match specified boundary displacements. Research efforts worldwide have been focused on optimizing these methods to reduce CPU cost while preserving the quality of the underlying mesh. Various strategies have been developed: for instance, [23] introduced a fast, accurate method based on Inverse Distance Weighting (IDW) that performs well for large displacements on complex 3D meshes; [24] further enhanced this method by integrating edge-swapping techniques, making

it suitable for very large deformations. To maintain mesh orthogonality, some researchers, such as [25], proposed methods that incorporate both translational and rotational components of the deformation, while combining kd-trees traversal and dodecahedron structures to reach high performance. In parallel, alternative strategies have been developed using Radial Basis Functions (RBFs) for mesh deformation. Coulier et al. [26] applied innovative methods such as Fast Multipole Method (FMM) to factorize and solve the RBF interpolation kernel. For structural analogy methods, recent works have been focused into solving linear elasticity equations using modern iterative solvers with better preconditioning. These solvers particularly aim to preserve boundary layer cells by locally increasing stiffness, as described in [27]. Additionally, conditions imposed on boundaries other than the aeroelastic interface often necessitate adjustments to the algorithms governing these methods.

In MIMAS, several mesh deformation techniques have been re-developed or improved compared to existing formulations in elsA. These techniques are:

- the Inverse Distance Weigthing (IDW) [28,29] which has been significantly sped up, extended to unstructured grids and made compatible with turbomachinery boundary conditions;
- a hybrid IDW and transfinite interpolation (TFI) [30–32] for structured grids which takes advantage of IDW acceleration;
- elastic analogy [27,33,34] which has been hybridized with IDW for parallelization.

The primary focus in MIMAS has been to accelerate the IDW deformation method, in particular because this method is well-suited for unstructured meshes and is used in conjunction with almost all other techniques. The cost of IDW increases rapidly with the fluid mesh size since it requires to compute $n \times m$ operations where m is the number of target points in the fluid mesh and n is the number of source points (located at the fluid-structure interface). The deformation vector $\widehat{D}$ is determined by calculating the inverse distance-weighted average of the source points as follows:

$$\widehat{D} = \frac{\sum_{i=1}^{n} w_i D_i}{\sum_{i=1}^{n} w_i}, \tag{1}$$

where D_i is the value of the displacements at the known source location i and w_i is the weight assigned to the known source location i usually defined as $w_i = \frac{s_i}{d_i^\alpha}$. Here, d_i is the distance between the current point location and the known source point i, α is a positive integer power parameter that controls the influence of the distance to the source and s_i is the local cell surface. The larger the value of α, the greater the influence of nearby points on the estimated value. To reduce significantly the cost of the brute force algorithm, two implementations were considered, both preserving the rotational component of the solid displacements. The first is based on a multi-layered clustering of source points, with directional damping to enforce boundary conditions. This method, illustrated in Figure 2, shows typical reduction factor between 5 and 10 compared to the brute force approach, for standard aircraft simulations. The idea is to regroup source points into clusters, and summing the contribution of the cluster barycentres only (instead of all points belonging to the interface). To make this method optimal, the cluster size is increased gradually with the distance to the source. Different layers $\mathscr{L}_k$ in the 3D mesh are defined, depending on their distance to the source, and each of them sees a different set of clusters. For instance, layer $\mathscr{L}_0$ close to the source sees all source points, $\mathscr{L}_1$ sees only half of the original set... The distance criterion that make switch from one layer to another is computed using the above formula:

$$\mathscr{D}_k = \sqrt{\frac{\alpha(\alpha+1)\sigma_k}{2\epsilon_t}}, \tag{2}$$

where α is the IDW exponent, σ_k is the average variance of coordinates and displacements inside the clusters associated with layer $\mathscr{L}_k$ and ϵ_t is a tolerance defined by the user on the error introduced by the IDW clustering approximation. Note that this is obtained by doing a Taylor expansion of the IDW formula assuming that the distances within the cluster are small compared to the distance to the source (see Appendix A). For target points close to the surface, a complementary strategy introduced earlier by Shepard is adopted. The latter selects only a patch in the surface with size proportional to the distance to the source [28].

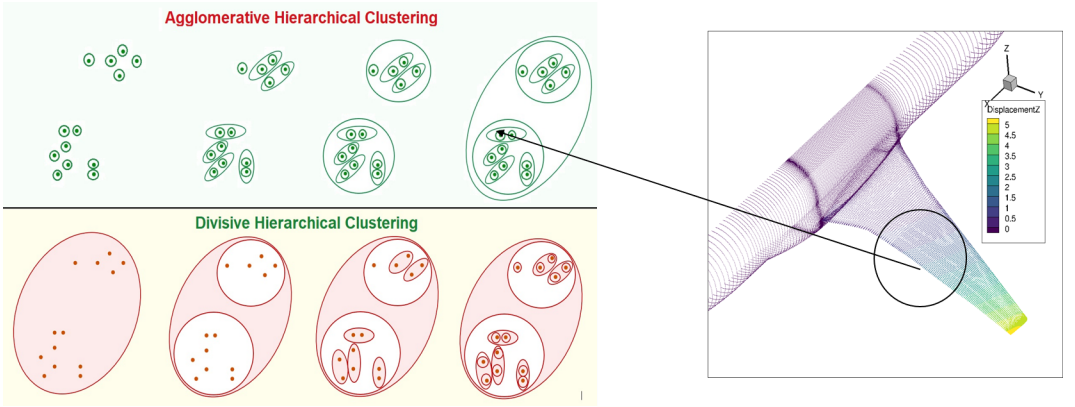


Figure 2. Illustration of the clustering of the source points for IDW cost reduction (adapted from N. Arya 2023, AI, Machine Learning and Deep Learning).

The second implementation of the IDW method in MIMAS is based on quaternion algebra [35] and Fast Multipole like algorithm (FMM) with recursive octree traversal. This algorithm, called *Quantum*, has proven to be efficient and has been tested on multiple configurations including icing simulations on airplane. For external flows in unconfined domains, it is generally faster than the clustering method, since the octree traversal combined with the FMM naturally eliminates interactions between distant points, in a continuous manner. Note however that for turbomachinery cases, involving confined domains, this method is not necessarily the fastest because boundary conditions necessitate several iterations. Moreover, the *Quantum* method is not highly scalable and is particularly hard to linearise due to the algorithm recursion and the fact that rotations are treated incrementally.

Results of these two IDW implementations are presented in Figure 3 with their MPI scalability using multi-cores architecture. Tests have been achieved on a turbomachinery fan, representative of the new generation of Ultra High Bypass Ratio (UHBR) engine. We superimpose the scalability curves of the hybrid TFI/IDW and elastic analogy methods (which have not been accelerated yet in MIMAS). Maximum number of cores used is 256 on the ONERA's development cluster but tests have also been carried out on ONERA's production cluster with up to 512 cores. The original brute force IDW method from elsA aeroelastic module lies above all externalized MIMAS methods. The first IDW algorithm with layered clustering (cyan line) scales quite well and is almost 8 times faster than the IDW from elsA (dashed blue line). The second IDW algorithm *Quantum* with FMM (red line) shows also consequent CPU time reduction (especially for external flows simulations, not shown here) but its scalability seems to be affected for a number of cores larger than 100.

Note that compact RBFs can also be used within MIMAS to deform a mesh and are solved using the MUMPs (MUltifrontal Massively Parallel Sparse direct Solver) library (see Section 2.4) but the choice of the appropriate kernel is not necessarily obvious and additional work is required

to clarify this point. Work is also in progress to make the elastic analogy compatible with unstructured meshes and to modernize the linear elastic solver (*move3d*), previously implemented in elsA for structured meshes exclusively. Finally, we stress that mesh deformation differs from adaptive re-meshing (such as AMR) since it preserves the connectivity and topology of the underlying mesh structure, as well as the number of cells in the mesh. When possible, mesh deformation is preferred to adaptive re-meshing because it avoids heavy connectivity computations. However, in the future, strategies combining both methodologies could be considered.

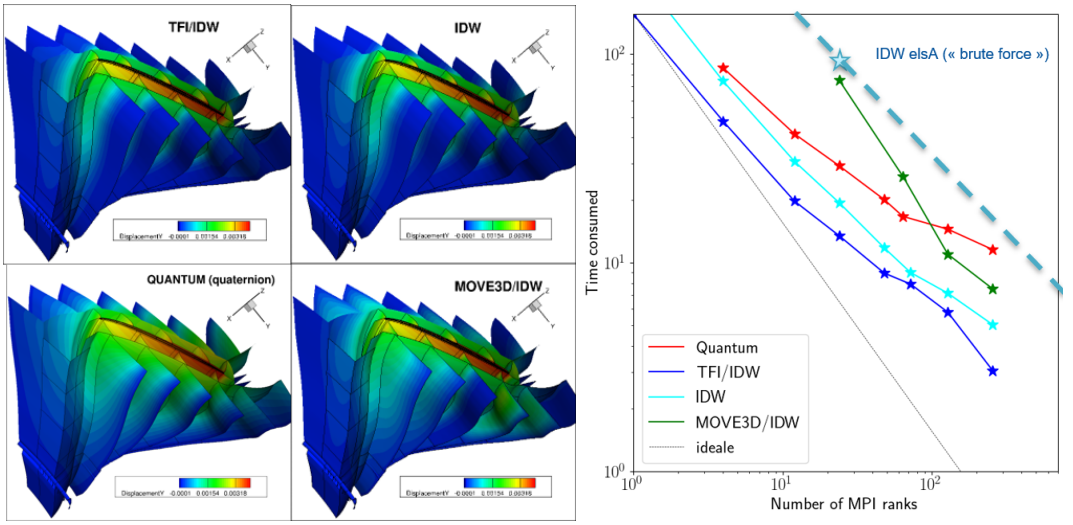


Figure 3. Left: Comparison of the azimuthal displacement field obtained through different mesh deformation methods available in MIMAS. Here they are applied on a single sector mesh of an UHBR turbomachinery fan. Right: Scalability of these methods on ONERA's development cluster for the turbomachinery case (4 million points). The cyan curve accounts for the layered clustering method with 8 layers and a 20 % tolerance on the error made on the IDW formula. The dashed line is the reference calculation using brute force algorithm in elsA.

2.4. *Parallel transfer algorithms*

Field transfer methods play a crucial role in the achievement of aeroelastic coupling. They allow to exchange physical information such as forces and displacement between the aerodynamic and structural grids. The main issue is that the fluid grid's numerical interface rarely matches with the structural elements boundary (see Figure 4). In some cases, the structural model may use simplified elements, like beams, which fail to fully represent the boundary shell, complicating accurate physical computations at the interface. So far, a wide range of algorithms have been proposed in the literature [36–42] and giving an exhaustive review on all these algorithms constitutes a full paper in itself. In most cases, transferring loads necessitates condensing the CFD force field onto a reduced set of points (compared to the original CFD grid) while the transfer of displacements often entails the opposite approach. Consequently, the algorithms ruling these transfers may vary considerably. The primary objective of these algorithms is to maintain high computational speed for large sets of interpolant surfaces while conserving physical quantities relevant to aero-structural interactions.

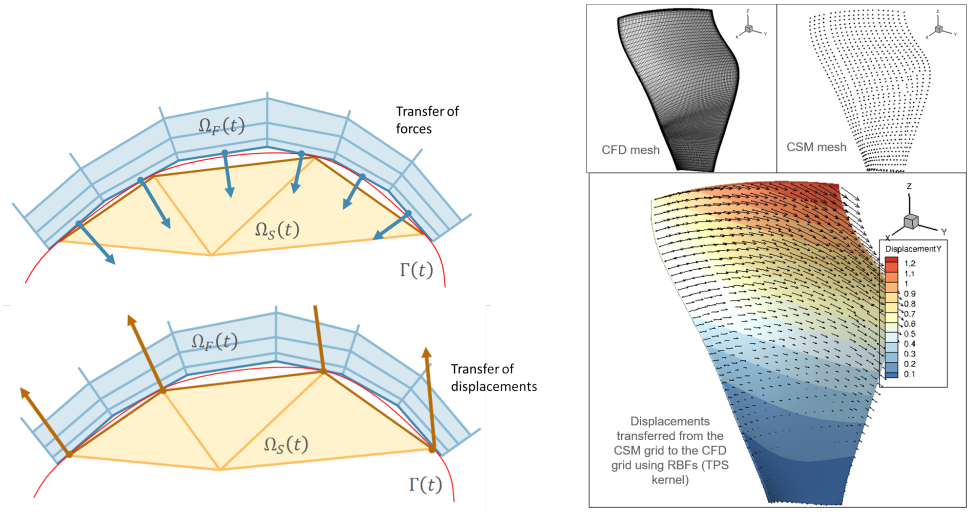


Figure 4. Left: Illustration of data transfer between the fluid and the structural grids. Right: Application of RBFs smoothing algorithm with TPS kernel on an Ultra High Bypass Ratio (UHBR) fan blade.

In MIMAS, we developed and upgraded several transfer algorithms (some of them originally present in elsA's aeroelastic module), with an emphasis on parallelizing them for efficient use in HPC environments. This parallelization was crucial to ensure these algorithms are scalable and capable of handling large datasets.

For the transfer of displacements (CSM grid to CFD grid), a first class of algorithms is the Radial Basis Functions (RBFs) smoothing. This interpolation method is rather convenient since it does not require the knowledge of the mesh connectivities, nor the structural discretization. Given a set of structural data (x_i, u_i) where x_i are the input points and u_i are the corresponding displacement values, the interpolated function $u(x)$ at a given point x in the space is expressed as:

$$u(x) = \sum_{i=1}^{N} w_i \phi\big(\|x - x_i\|\big) + P(x), \tag{3}$$

where: N is the number of radial functions which is chosen to be equal to the number of known structural source points; ϕ represents the chosen radial basis function, which can be "thin plate spline" (TPS), gaussian, multiquadric, etc.; $\|x - x_i\|$ is the distance between the chosen point x and the known data point x_i, and w_i are the weights associated with each data point. Determining the weights involves solving a linear system of equations of size N, typically represented as a dense matrix equation. The weights w_i are adjusted to cancel the error between the interpolated values and the actual data. The decomposition in radial basis functions is often modified to also include polynomial term

$$P(x) = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 z + \cdots \tag{4}$$

to ensure conditionally positive definite radial functions (e.g. TPS), together with matching constraints on the expansion coefficients [41,43,44]. To sum up, the interpolation problem is equivalent to solve for the system:

$$\begin{bmatrix} \mathbf{\Phi} & \mathbf{P} \\ \mathbf{P}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \boldsymbol{\beta}^T \end{bmatrix} = \begin{bmatrix} \mathbf{u} \\ 0 \end{bmatrix} \tag{5}$$

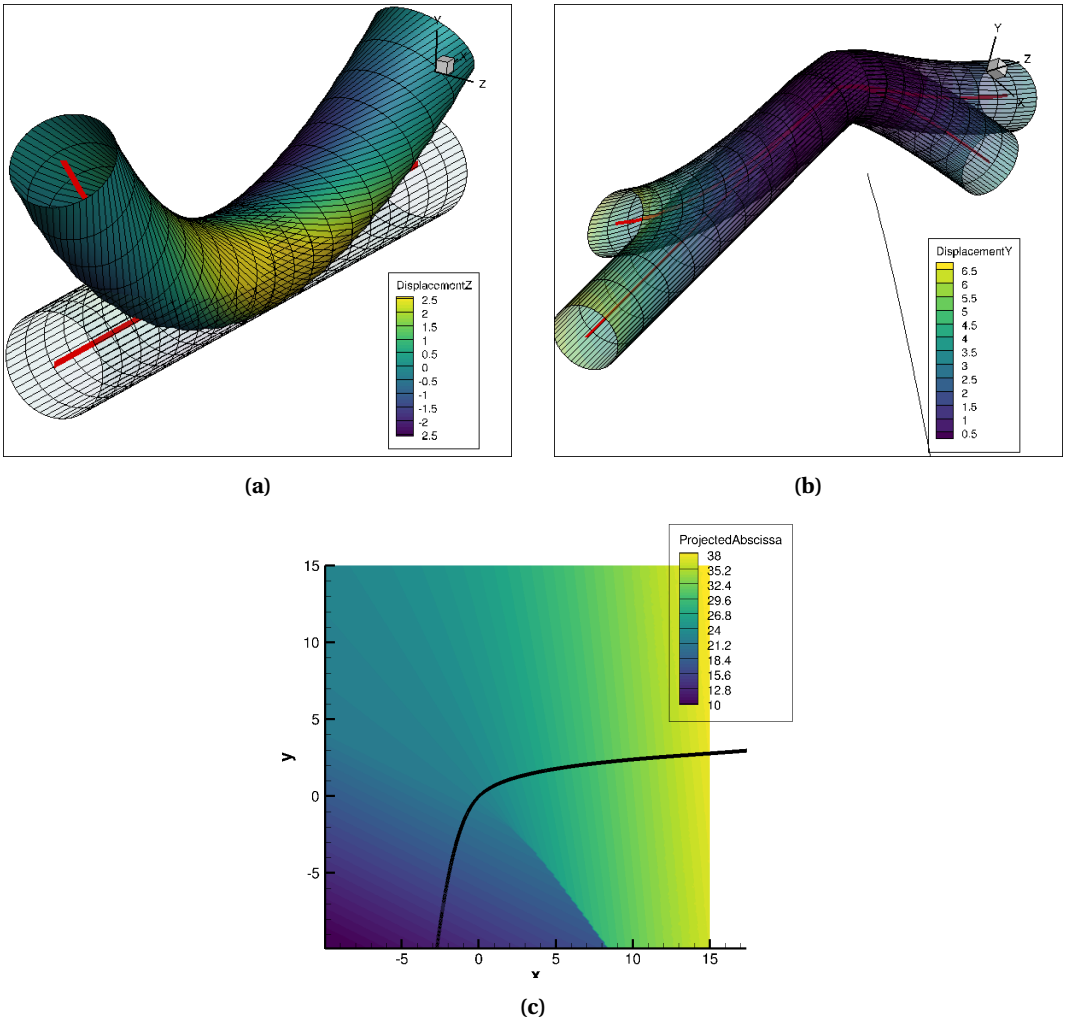
with $\mathbf{\Phi}_{ij} = \phi\big(\|x_i - x_j\|\big)$, $\mathbf{w} = (w_1, w_2, \ldots, w_N)$, $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \beta_3)$ and

$$\mathbf{P} = \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & y_N & z_N \end{bmatrix}. \tag{6}$$

The major drawback of this method is its cost when the structural model becomes large ($\gg 10^4$ points). If used in the context of mesh deformation, this is even worse since the size of the matrix can exceed easily 10^5 by 10^5 points. In the literature, several techniques address this issue while maintaining a smooth global displacement field [40]. In the legacy elsA aeroelastic module, a pre-processing step involved a manual selection of a reduced set of structural points for transferring the displacements. The selection was performed in order to improve the numerical stability of the transfer method (e.g. avoid the excessive point density that could yield ill-conditioning interpolation matrices when the RBF are used) and to conserve the geometric properties of the structural deformations (e.g., displacement gradients, curvature zones). MIMAS offers an automatic technique to sub-sample the structural model using agglomerative clustering methods, similar to those employed for mesh deformation (see Section 2.3). These methods naturally tend to select points near sharp gradients and they give satisfactory results for turbomachinery blades. However, they have limitations, as they rely solely on spatial proximity or field distribution and do not consider the mechanical properties of the points. A dedicated parallel LU solver for dense matrix, using the Message Passing Interface (MPI) protocol, has been developed to solve the linear problem (5). Another solution provided by MIMAS is to use compact support functions, which can result in a sparse RBF matrix. One advantage of this method is that it effectively suppresses the influence of distant points, which is particularly beneficial when dealing with a wing and a horizontal tailplane for example. This localized influence helps in some cases in preserving the physical behaviour of the problem. To provide with an efficient computation of the sparse system, we linked MIMAS with the MUMPs library [45], which is considered today as one of the most efficient direct solver for sparse linear system.

A second class of algorithms is based on beam kinematics. In that case, source data are known along a beam line and the goal is to reconstruct the displacement around it. Given a point P in space, the algorithm first computes its orthogonal projection onto the beam line and reconstructs the displacements locally using hermite spline elements associated with the beam. The novelty in MIMAS lies in the ability to account for significant deformations, automatic detection of corners and broken lines, and the removal of degeneracy in orthogonal projection. Additionally, it lifts the restriction associated with Euler–Bernoulli hypothesis (where it is assumed that sections remain orthogonal to the beam curve), while still maintaining orthogonality in projections. Examples of deformation fields using this technique are illustrated on Figure 5. The red lines account for the beams from which the deformations are known, while the surface meshes around are the target grids where displacements are re-constructed.

Finally, for load transfer, two classes of algorithms have been implemented: the first is based on a nearest neighbour search and conserves the resultant of forces and moment. The structural points for transfer are chosen on the key load-bearing elements, such as beams, wing box, and other primary support components of the structure, so that forces are correctly absorbed by the whole structural model. The main benefit of this method is that it is quite straightforward and cheap in terms of computational time. However when important change in the geometry occurs or when pressure loads encounters huge gradient (typically near the leading or trailing edge of an airfoil, or near the shocks), this approach can lead to a wrong local distribution of forces on the structure. In addition, it does not conserve energy, and methods using virtual work technique can be preferred in that case. The second method, based on virtual work conservation, uses



(a) **(b)**

(c)

Figure 5. Illustration of kinematics beam transfers. (a) An application of the transfer around a flat beam line with combined torsional and bending motion. (b) The deformation around a curved beam line with critical degenerated surface for orthogonal projection. (c) The intersection of the critical surface with the xy plane (thick black line).

Radial Basis Functions (RBFs) for interpolating forces and displacements. A base of continuous functions $\vec{\Phi}^i$, $i = 1,\dots,N_b$, is chosen so that an "admissible" displacement field $\overrightarrow{\delta u}$ can be decomposed into:

$$\overrightarrow{\delta u}(x,y,z) = \sum_{i=1}^{N_b} w_i \vec{\Phi}^i(x,y,z). \tag{7}$$

The conservation of virtual work between fluid and structure associated to each individual basis function leads to:

$$\langle \vec{f}_{\text{aero}}, \vec{\Phi}^i \rangle = \sum_{j=1}^{N} \vec{f}_{\text{aero}}[j] \cdot \vec{\Phi}^i[j] = \sum_{k=1}^{M} \vec{f}_s[k] \cdot \vec{\Phi}^i[k] = \langle \vec{f}_s, \vec{\Phi}^i \rangle \tag{8}$$

where j and k are the indices of the fluid and structural nodes respectively, N and M are the number of aerodynamic and structural nodes. Since the aerodynamic nodal forces $\vec{f}_{\text{aero}}$ and the basis functions are known, the first term can be easily evaluated. In the right-hand side, only the terms $\vec{f}_s[k]$ are unknown. The basis functions $\vec{\Phi}^i, i = 1,\dots,N_b$ is constructed from a sum of Radial Basis Functions (RBFs) and enforce unitary displacement at control point and zero elsewhere. These functions are supplemented with a polynomial, with constant that ensures the conservation of global loads and a linear term to preserve global moments. Efforts are preserved locally through the local support of the RBF functions. The linear system for the unknown $\vec{f}_{s_k}$ is solved using the same methods described previously for displacement transfer. Note that this method can be significantly costly since we need to solve undetermined weights for each basis function $\vec{\Phi}^i$, $i = 1,\dots,N_b$. It gives generally smoother results than the nearest neighbour (outside discontinuities such as shocks), but can lead to artificial loads concentration when structural elements intersect.

2.5. *Multiple FSI interfaces*

Systems like rotorcraft blades, wings with control surfaces, or compressor stages in turbomachinery involve different structural parts that may be numerically treated separately for several reasons (physical behaviour, performance...). Multiple FSI interfaces allow each structural component to be treated independently in terms of transfer algorithms, while interactions between components are captured exclusively through the fluid dynamics. However, a key challenge of this approach is that transferring individual displacements from each component to its corresponding aerodynamic surface create discontinuities between adjacent surfaces. A well-known example is the interaction between the fuselage and the wings of an aircraft. Suppose that the wing and fuselage are solved using the same structural model, but both components are decoupled during the transfer process. The interpolation methods applied to each component do not process identical data, resulting in discrepancies at the junction between the two. To solve this issue, we implemented in MIMAS an algorithm allowing to match the different components. Consider two non-overlapping sets of points S_1 and S_2 for each structural component and their corresponding aerodynamical surfaces A_1 and A_2. First two transfers are achieved on $A_1 \cup A_2$, the first using the S_1 structural data, the second the S_2 data, giving respectively a displacement field u_1 and u_2. Using efficient kd-trees, we compute for each aero-dynamical point its distance d_I from the intersection between the set A_1 and A_2. This is estimated by computing the distance to the nearest neighbour belonging to the opposite surface. Note that better strategies involving convex hull border or surface fitting with implicit function evaluation may provide more accurate distance to the intersection. When the distance of the aerodynamical points from the intersection is known, we use blending functions (such as sigmoids or polynomials) to match the displacements between the two surfaces. For example, if we use a cubic blending between A_1 and A_2, the transferred displacement is:

$$u(x,y,z) = w u_1(x,y,z) + (1-w)\,u_2(x,y,z) \tag{9}$$

with

$$w = \begin{cases} \left(\frac{d_I}{\delta}\right)^3 - \frac{3}{2}\left(\frac{d_I}{\delta}\right)^2 + 0.5, & \text{if } d_I \le \delta, \\ 0, & \text{if } d_I > \delta. \end{cases} \tag{10}$$

Here δ represents the depth into the neighbourhood region and is set to the distance of the furthest point identified during the k-nearest neighbour search.

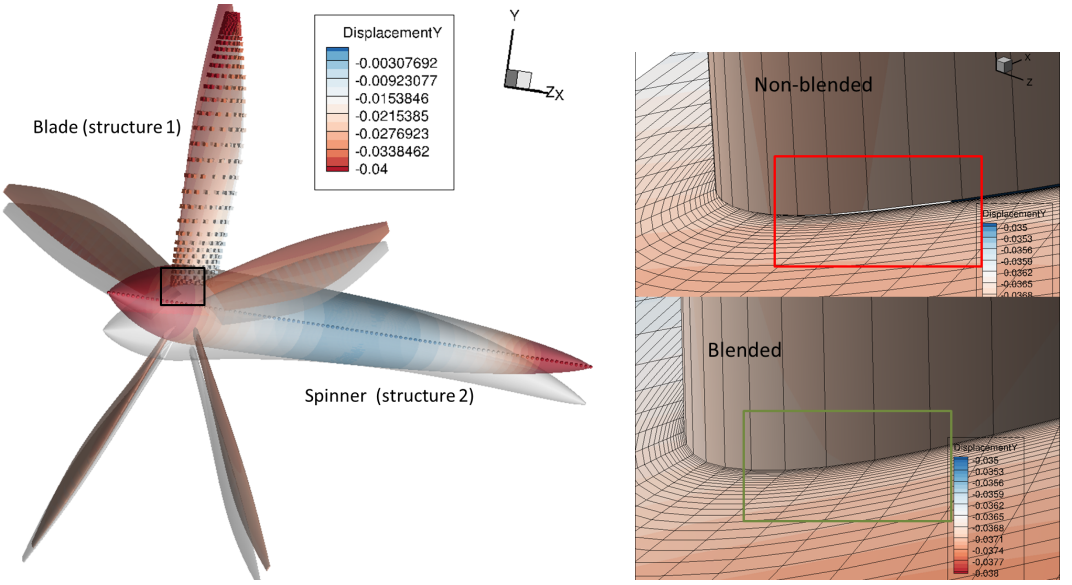


Figure 6. Illustration of the blending method for multiple FSI interfaces when independent transfer algorithms are used for each structural component. On the left is shown the deformation imposed on a 5-bladed propeller with bending of the spinner and bending-torsion of the blades. The cube points on the blade represent the structural nodes used for the transfer on the propeller blades while aligned spheres are the structural nodes for the spinner axis. Right top: Displacement field and fluid mesh after the transfer without blending. Right bottom: Same after blending.

An illustration of the method is provided in Figure 6, where it is applied to a 5-bladed propeller. The structural displacements are modelled using a global analytical mode (though not necessarily physically realistic) that combines the bending of the spinner with the bending-torsion motion of the blades. While the structural components are computed consistently, the displacement transfer between the blade (S_1) and the spinner (S_2) is handled separately. For the blade, we employ a Radial Basis Function (RBF) approach based on a set of 3D points on the blade surface, represented by cube points in Figure 6. For the spinner, only points along the X-axis are considered, with no point taken from the spinner's surface. A beam kinematics transfer method is used to model the displacements on the fluid surface, which is shown in color in the left panel of Figure 6. Once the two independent transfers are combined, we observe that the displacement field is discontinuous between the spinner and the blade, as expected. This discontinuity results in a non-matching intersection, as illustrated in the top right panel of Figure 6. After applying the blending algorithm using kd-tree search, we demonstrate that the displacements become smooth in the intersection regions, with the two meshes coinciding, as shown in the bottom right panel.

Note that there are cases where we would like to decouple FSI interfaces that are not directly connected, such as a wing and a horizontal tailplane. In such cases, we can blend each surface with the shared connecting surface (i.e. the fuselage) or use compact radial basis function (RBF) interpolation. The latter naturally decorrelates the two surfaces once a certain distance condition is satisfied.

2.6. *Coupling strategies and time-marching*

The standard approach in coupling fluid/structure is a sequential execution of fluid and structure solvers. This approach, although known to be quite robust, generally leads to slower convergence and load imbalances in a parallel setting. With the modular and external approach of MIMAS, new ways of coupling fluid and structure can be explored, unlike elsA, which operates within a more rigid and static architecture. First, for static problems, we are not anymore limited by the fixed-point approach with constant relaxation parameter and constant coupling period. We can indeed call the structural code at chosen fluid iterations to optimize convergence. A criterion can be established, such as setting a threshold for the relative change in the fluid residual compared to the previous coupling iteration. The relaxation parameter can also be prescribed by users at any time. Other strategies, like Aitken under-relaxation [46] have been used and tested within our new framework (see Section 4.1). This algorithm adapts the relaxation factor at every iteration based on current and previous state of the structure. If we define the residuals of the structural displacements at previous iteration $k-1$ and current iteration k,

$$\mathbf{r}_{k-1} = \mathbf{u}_{k-1} - \mathbf{u}_{k-2} \qquad \text{and} \qquad \mathbf{r}_k = \mathbf{u}_k - \mathbf{u}_{k-1}$$

and $\Delta\mathbf{r}$ the residual difference:

$$\Delta\mathbf{r} = \mathbf{r}_k - \mathbf{r}_{k-1}.$$

Then the relaxation factor at current step ω_k is calculated as

$$\omega_k = -\omega_{k-1} \frac{\mathbf{r}_{k-1} \cdot \Delta\mathbf{r}}{\|\Delta\mathbf{r}\|^2}$$

where the dot product here represents the sum of the individual product on each structural degree of freedom. To calibrate the initial relaxation parameter ω_0, we use a similar approach inspired by the preCICE library [12]. For dynamical problems, it is almost similar, except that the fixed point approach is applied at each physical time-step. Exchanges between the fluid forces and the structural displacements are performed during sub-iterations (dual loop) of the time integration method (see Figure 7). In elsA, the strategy was slightly different since the aeroelastic solver iterates multiple times on the same time step to converge the fluid and structure. We emphasize though that for dynamical problems, better strategies involving mid-point time schemes, asynchronous coupling schemes or parallel quasi-Newton least-squares solvers [12], mixing both fluid and structure states, are possible. Research on these coupling time-scheme is active at ONERA and could replace progressively the current implementations.

Another scope of research is the externalization of non-linear iterative fluid or structural solvers themselves. Over the past decade, significant efforts have been made to externalize the Jacobian and its matrix-vector products from both CFD and CSM solvers. This is particularly useful if one wants to solve the implicit phase externally. In particular, this offers the capability of pre-conditioning the system with coupling terms and use more advanced or dedicated Newton–Krylov solvers for FSI problems. Ultimately, the monolithic approach could be investigated, for which the whole fluid/structure system is solved externally. Note that other strategies such as the Time Spectral Method (TSM) for periodic systems are also investigated but remain out of the scope of the MIMAS library for now. More details about their current implementation at ONERA is presented in [11].

2.7. *Non-linear structural coupling*

As we mentioned earlier in the introduction, high-aspect-ratio wings and large-diameter fans or propellers exhibit increasingly nonlinear behavior due to large displacements. Another cause of nonlinearity is the friction between structural components (e.g. for blade-disk assemblies)

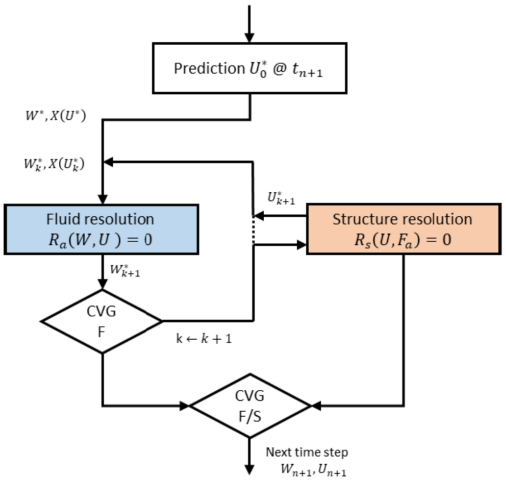


Figure 7. Illustration of the coupling scheme for dynamical problem in MIMAS. The structural displacements and the fluid mesh are updated regularly during the dual loop of the "Gear" or "DTS" (Dual Time Stepping) temporal scheme.

which can influence significantly the aeroelastic response of aeronautical structures. A nonlinear modelling of the structure is therefore required for the resolution of the coupled aeroelastic problem. When structural nonlinearities due to large displacements U are considered, the dynamic equation of motion contains an additional term $f_{\text{nl}}(U)$:

$$M\ddot{U} + C\dot{U} + KU + f_{\text{nl}}(U) = F_a(U,t)$$

where M and C are the mass and viscous damping matrices and $F_a(U,t)$ the aerodynamic forces acting on the structure. In the static case, the equation reduces to

$$Ku_s + f_{\text{nl}}(u_s) = F_{a,s}(W)$$

where u_s is the displacement resulting from a steady aerodynamic force. The treatment of the nonlinear term requires an iterative Newton process. In the dynamic case, the Newton algorithm is combined with the time integration scheme, generally handled with Newmark or more generally HHT methods for structural problems. Dealing with such nonlinear structural models within the CFD solver kernel is not necessarily the most appropriate, as different methods and solvers than those used for the fluid are required. The modular and partitioned approach is then well suited to rely on an external structural solver. In MIMAS, an interface has been implemented with the finite element code NASTRAN and preliminary developments have been carried out to couple MIMAS with Code_ASTER of EDF. Upcoming developments are also intended to couple with the next generation ONERA code A-set. For NASTRAN we use the SOL400 nonlinear solution enabling large displacements, and the coupling is performed either by files or through the C++ OpenFSI interface. This interface offers a way to stop and restart NASTRAN at each coupling iteration, and communicate buffers address directly between NASTRAN and the Python driver. Such a coupling with OpenFSI has already been performed in the past at ONERA, using elsA's legacy aeroelastic module, enabling a partial externalization [47]. Note that user Python interface of MIMAS allows to prescribe several NASTRAN parameters to calibrate the non-linear resolution.

The structural non-linearities could be modelled within a projection based by a Reduced Order Model (ROM). This can be useful for dynamic aeroelastic computations to avoid the coupling with the full finite element model at each physical iteration. In projection based

reduced order models, the displacements of the structure are computed by using a representative projection basis that captures the non-linear structural response, so that $U \approx V q$ with $V = [\Phi, D]$, where Φ represents the linear eigenmodes and D represents additional modes (e.g., modal derivatives or dual modes). The nonlinear dynamic equations of motion are then projected onto this reduced basis, resulting in a simplified set of scalar nonlinear equations:

$$\mu\ddot{q} + \beta\dot{q} + \big(\gamma + \gamma_{\mathrm{nl}}(u_s)\big)q + \widetilde{g}_{\mathrm{nl}}(u) = f_{a,g}(W).$$

With the classical approach, the nonlinear internal forces need to be assessed by means of a FE solver, reducing the time efficiency of the reduced order model. Thus, an autonomous reduced order model could be obtained by considering that the internal non-linear forces are approximated by a third-order polynomial function of the generalised displacements. The k-th coefficient of the nonlinear force vector is thus:

$$\widetilde{g}_{k,\mathrm{nl}} \approx \sum_{i,j} \rho^k_{ij} q_i q_j + \sum_{i,j,m} \theta^k_{ijm} q_i q_j q_m.$$

The polynomial coefficients are identified from a set of precomputed nonlinear static solutions using the Implicit Condensation method with Expansion (ICE) [48]. This provides an explicit expression of the reduced-order model in terms of the generalized coordinates q, which can be efficiently coupled with the fluid solver. The nonlinear reduced order model detailed above has been coupled to the fluid solver elsA on the simplified test case of a flexible beam placed in the wake of a cylinder generating an unsteady forcing [49]. It has also been used to evaluate the dynamic response of a UHBR fan blade subject to an external forcing induced by an inlet distortion [50]. These studies were conducted at ONERA outside the MIMAS framework, but more recently, reduced-order model of the ERATO helicopter blades have been coupled with fluid models within the MIMAS framework (see Section 4.3). For that, an entire interface has been written between MIMAS and an external tool designed to build ROMs from finite element solver Code_ASTER (EDF).

2.8. *Coupling with advanced HPC fluid codes (CODA, SoNICS)*

MIMAS handles a generic interface to HPC CFD solvers which is based on class abstraction. The class provides methods to achieve basic operations, including advancing one or several fluid iterations, extract forces and update geometry. On the ONERA/SAFRAN side, the SoNICS code is a next-generation software designed to enhance performance and usability in turbomachinery applications. It features a new architecture that incorporates operator graph structure for tailored calculations, cache-blocking, automatic linearized and adjoint through the Tapenade library [51], efficient code generation for optimized hardware usage (including GPUs), and advanced capabilities for handling complex physics. SoNICS handles multiple fluid species and dynamic mesh adaptation. We recently carried out developments within the SoNICS code to implement the Arbitrary Lagrangian Eulerian method for enabling grid velocities for aeroelastic studies. We also wrote the triggers classes to achieve basic operations mentioned earlier such as updating the grid and extracting forces. In particular, updating the grid in SoNICS has required to modify the primary execution graph to propagate the new metrics everywhere in the code and to each operators. In terms of data, it utilizes the standard CGNS format, featuring polyhedral connectivity representations (NGon). First simulations have recently been conducted with harmonic forced motions on the M6 wing configuration to validate the implementation of the ALE formulation for deforming grids.

On the ONERA/DLR/Airbus side, CODA is an advanced computational code which includes a large variety of spatial schemes (finite volumes & discontinuous Galerkin). It focuses on optimizing aircraft design through efficient numerical methods, operators linearization, and integrates

various physical components, including fluid-structure interactions. It relies on the FlowSimulator DataManager (FSDM) which provides an HPC library for CFD-based simulation workflows, models, data manipulation and multiprocessing. As the core library includes Python interfaces, data is easily accessible through MIMAS. However, since the data format is very different from the standard CGNS format, an important work has been performed to convert efficiently the mesh and the aeroelastic data (forces, displacements, etc.) into our own representation. Section 4.1 shows first aeroelastic calculation using CODA and MIMAS jointly, with comparison with elsA using the same mesh.

2.9. *Coupling with Vortex Particle Method (VPM)*

Finally, MIMAS is able to couple with medium fidelity method such as the Vortex Particle Method (VPM) [52]. The originality of such coupling is that the fluid model is generally mesh-free. In VPM, the flow is represented as discrete particles that carry vorticity. Each vortex particle has properties such as position, circulation strength, and sometimes additional attributes like velocity or density. The method operates in a Lagrangian framework and is particularly efficient to simulate complex wake dynamics. The governing equation is the transport equation of the vorticity, which includes a stretching term and a viscous diffusion term. The viscous diffusion term is used to model the influence of both the molecular viscosity and the turbulence associated to small eddies. In MIMAS, we implement an interface with ONERA's code Vulcains [53]. One of the existing formulations, consists in a lifting-line/VPM coupling where the lifting components (e.g. blades, wings...) are modelled with the Blade Element Theory (at the quarter chord position). At each time step, new particles are generated based on this lifting-line model. The local velocity induced by the particles is then projected onto the normal plane of each lifting-line section, from which the angle of attack, Mach number, and Reynolds number are derived. Using 2D airfoil tables and Kutta–Joukowski theorem, it is then possible to calculate the circulation for each blade section. New particles are generated to account for this circulation and an inner iterative loop is performed at each time step to ensure convergence between the newly generated particles and the blade circulation (as the new particles also contribute to the velocity induced along the lifting line). For aeroelastic coupling, there are two different scenarios:

(i) the structural model is a beam model which coincides with the lifting line; in that case transferring structural forces and displacement between the fluid and the structure is trivial;
(ii) the structural model is based on 3D or 2D finite elements; then the full pressure and force field are reconstructed on the surface using the 2D airfoil tables.

The transfer of the displacements can be done using RBFs on an auxiliary grid (though not optimal and physically questionable) or directly by condensing the structural displacements into a local rigid motion (translation/rotation) for each sections of the lifting line model. An example of simulation coupling VPM and nonlinear Finite Element Model (FEM) is presented in Section 4.3. Note that the VPM can be also coupled with Eulerian grid model or URANS simulations but this is out of the scope of the present article.

3. Validation cases: classical aeroelastic calculations

Before presenting applied cases that showcase new capabilities, we first demonstrate that MIMAS can accurately replicate standard aeroelastic calculations which have been historically computed using elsA's aeroelastic module [14].

3.1. *Static coupling*

To validate the modular approach, we first test the case of a static coupling on the Common Research Model (CRM) plane configuration. To compare MIMAS with elsA's aeroelastic module, a basic fixed-point method, including constant relaxation of the structural displacements, is used. In MIMAS, fluid iterations are solved externally by elsA while the structural problem is solved directly at the Python level using a condensed structural stiffness matrix with smaller dimension compared to the initial NASTRAN model. The model comprises mass loads for the engine and the fuel, while gravity force is added externally by MIMAS using the area/thickness of shell elements. The fluid domain is discretized with a coarse structured grid of about 1 million cells. The CFD resolution is inviscid (Euler model) and uses a Roe scheme with the Van Albada flux limiter. Fifteen fixed point iterations between the fluid and the structural solvers are performed to reach the equilibrium. The coupling is handled with the TFI/IDW hybrid method for deforming the structured mesh, and the nearest neighbour and RBFs respectively for force and displacement transfers. The points for the transfers are represented in the right panel of Figure 8. A Thin Plate Spline (TPS) kernel is used for RBFs, which consists in $r^2 \log(r)$ functions, and the interpolants are computed with our dedicated MPI LU solver. Note that for transferring loads to the structure, we exclude forces acting on the fuselage and tail.

Comparison between the computations with MIMAS and elsA's aeroelastic module are shown in Figure 8 for a Mach 0.85 and an angle of attack of 1.32° at altitude 36 000 feet. The pressure distribution for this operating point obtained by MIMAS is also shown in the left panel. The agreement between both computations is excellent in terms of vertical displacement, although the mesh deformation (TFI/IDW method) and transfer methods are not exactly implemented in the same way. We check also that the forces transferred to the structural grid are very similar. Note that with MIMAS, this calculation has been improved to take into account the full finite element model, using the nonlinear NASTRAN solver (SOL 400). For the selected flight condition considered here, no significant differences are however observed with respect to the statically condensed linear model.

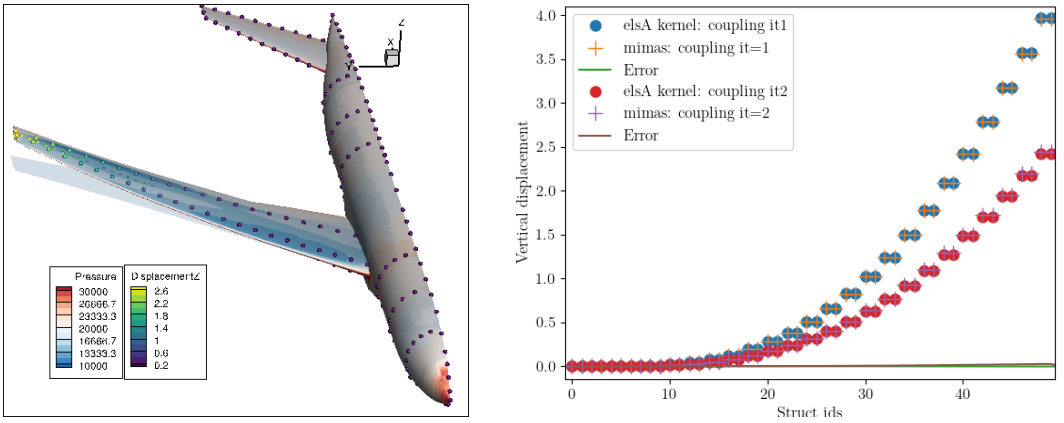


Figure 8. Illustration of the CRM static coupling. Left: Pressure field on the plane with condensed structural points (force and displacements nodes). These points are coloured with the vertical displacement. Right: Comparison of the vertical displacement along the span between elsA's legacy aeroelastic module and MIMAS at first and second coupling iterations.

3.2. *Harmonic forced motion*

Validation of the modular approach has been performed in the unsteady case for forced harmonic motion simulations. The test case is a state-of-the-art fan blade representative of a modern Ultra High Bypass Ratio (UHBR) engine, presented in Figure 9. In these simulations, a natural mode of the structure is excited at given frequency(ies) with a relatively small amplitude to characterize the aerodynamic response and ultimately the generalized aerodynamic forces. The linear unsteady aerodynamic response is computed using CFD codes (such as elsA) and can be re-injected into a structural linear solver to assess the aeroelastic stability (useful for flutter prediction). Mesh deformation is generally computed once, during a preprocessing step stage, and the corresponding mesh deformation associated to the modal shapes is interpolated in time using a sinusoidal law. In the modular approach, the coordinates of the deformed mesh, as well as the grid velocity are calculated outside the CFD code.

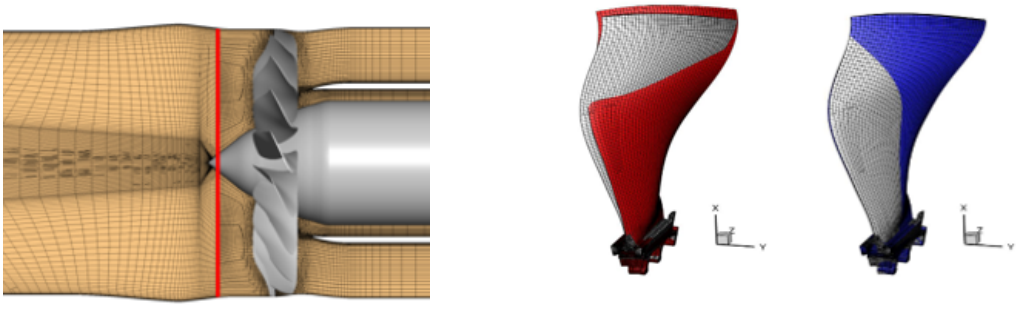


Figure 9. Ultra High Bypass Ratio (UHBR) fan blade test case and real/imaginary mode shapes of interest (deformation artificially amplified for visualization).

For the UHBR fan, a modal basis for the structure is first pre-computed using SAMCEF. Data of the modal shape (displacements) are transferred from the SAMCEF grid to the CFD grid by MIMAS using Radial Basis Functions during the pre-computation phase. We then select one particular mode (here the first bending mode 1F-0D with zero dephasing between the blades) and apply small oscillations to this mode with a given frequency, modal amplitude of 0.3 mm and initial phase of $\pi/2$. Note that with the new mesh deformation algorithms implemented in MIMAS based on IDW method, amplitude up to 4 mm in azimuth of the fan-blade can be reached, which is almost two times the tip gap size (between the shroud and the blade). The slipping condition on the shroud is handled with a directional damping implemented specifically for IDW.

For MIMAS, we use again elsA as an external solver for the CFD calculation. Only one sector is simulated in order to save computational time and periodic conditions are used in azimuth. The mesh is composed of approximately 3.2 million points, divided into 48 blocks. Note that modal shapes with dephasing between the blade can also be simulated in the modular aeroelastic environment, using only one sector. For that, a complex representation of the mode is used, in addition to the so-called phase-lagged condition in elsA. MIMAS mesh deformation module is able to deal with such complex deformation. Figure 10 (left) shows the aerodynamic forces projected onto the structural mode of vibration (GAF), first computed with the aeroelastic module of elsA (blue) and second computed with MIMAS (orange). Clearly, both solutions match perfectly, which validates our modular approach. Note that the same preprocessed deformed mesh for the modal shape has been used in both calculations using the elastic analogy method (*move3d*). With the modular implementation, other mesh deformation methods are

available and have been compared in this context. The phase portrait (Lissajous) of the GAF is compared on the right panel of Figure 10 for the last cycle of vibration: a relatively good agreement is obtained when using TFI, IDW or the quaternion approach (Quantum), although it is interesting to notice slight variation of the Lissajous sizes when the deformation algorithm changes. Variations up to 15 % can be measured on the aerodynamic damping of the mode according to the deformation method.

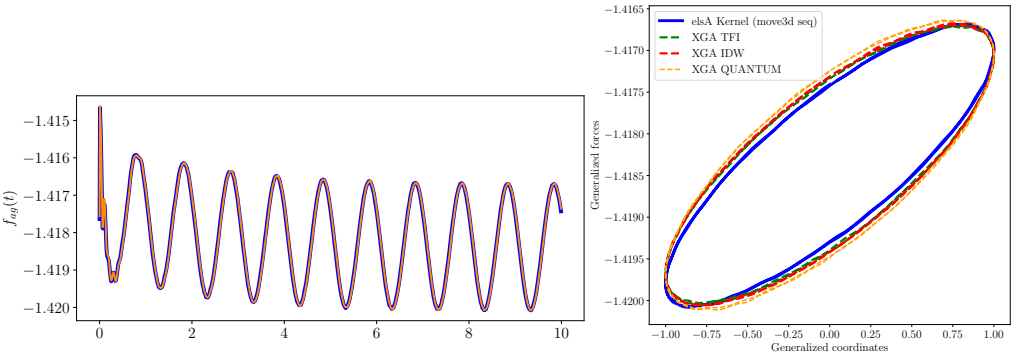


Figure 10. Comparison of elsA's aeroelastic module and MIMAS modular solution for forced motion using the same deformed mesh (left) or alternative external mesh deformation algorithms (right). On the left panel are shown the generalized aerodynamic forces as a function of time. On the right, are shown the Lissajous curves (phase portrait) of the aeroelastic response.

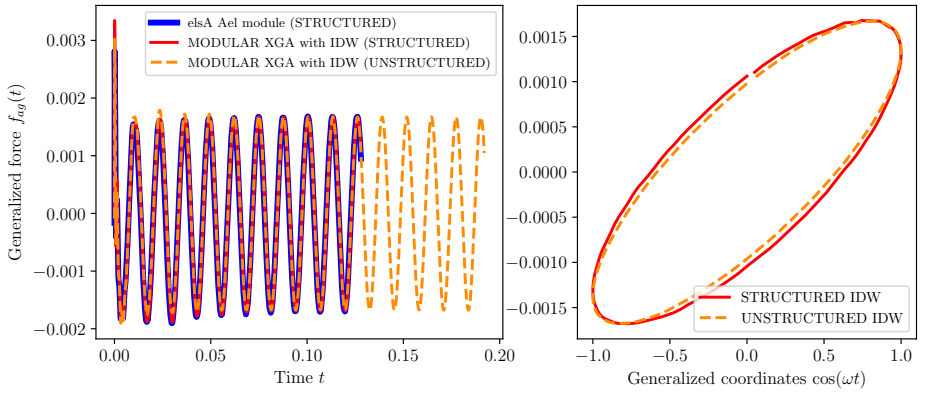


Figure 11. Comparison of forced motion aeroelastic computation with structured or unstructured (destructured) mesh.

Another key advantage of MIMAS compared to elsA's aeroelastic module is the ability to conduct aeroelastic simulations on unstructured grids. In this example, the structured grid was "destructured" for comparison purposes, using the same hexahedral mesh but treated as unstructured. The results match relatively well (see Figure 11) between the two grid representations. Here, the IDW mesh deformation method was applied since the structural analogy approach has not yet been implemented in the modular framework for unstructured meshes.

3.3. *Dynamic coupling with linear structure*

Finally, we demonstrate that MIMAS can reproduce fully coupled dynamical calculations under the assumption of a linear structure, a capability that was previously achievable with elsA aeroelastic module. In this type of simulation, the linear dynamic behavior of the structure is represented by a limited set of eigenmodes $\Phi = [\phi_1, \ldots, \phi_r]$. The displacement is thus approximated as a linear combination of these modes $U = \Phi q$ and the dynamic equation of motion is simplified through projection into r equations:

$$\mu\ddot{q} + \beta\dot{q} + \gamma q - f_{\text{ag}}(W) = 0.$$

Here, q are the unknown generalized coordinates, μ, β, and γ represent the generalized mass, damping, and stiffness matrices, respectively, while $f_{\text{ag}} = \Phi^T f_a$ denotes the vector of generalized aerodynamic forces. This set of equations is solved externally in MIMAS using a Newmark algorithm for the generalized coordinates q, from which the physical displacements can be derived. As mentioned in Section 2.6, the coupling procedure involves mechanical steps and mesh deformation in the implicit dual cfd loop.

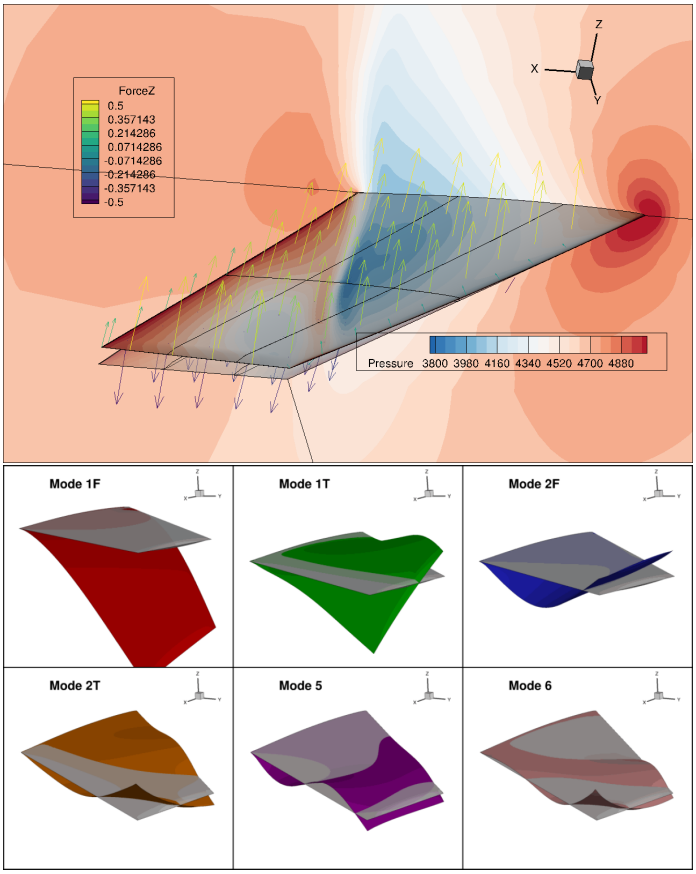


Figure 12. AGARD [54] dynamic coupled simulation — Pressure distribution and deformation of the wing at $t = 3T$ (top) and the six first modes shapes used for projecting the dynamics (bottom).

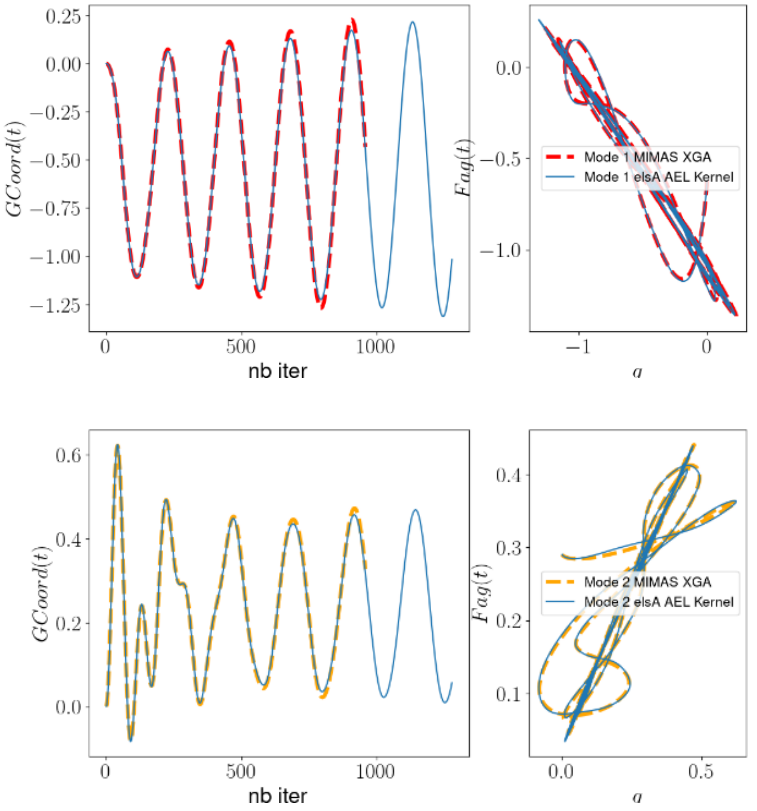


Figure 13. AGARD dynamic coupled simulation — Comparison of the dynamic response of the first two structural modes between MIMAS calculations (dashed lines) and elsA AEL module (plain lines). On the left panel are plotted the generalized coordinates as a function of physical time iterations. On the right panel is shown the phase portrait of the dynamical motion (generalized force vs. generalized coordinates).

For validation, we run a coupled simulation of the AGARD 445.6 wing configuration using either MIMAS or the elsA aeroelastic module. We adjusted the structural model to exhibit an unstable behavior. The flow is inviscid and transonic (Ma = 0.96, AoA = 1°), while the dynamic of the structure is modelled using the first six eigenmodes. The shapes of the modes are illustrated in the bottom panel of Figure 12. The simulation is started from a steady equilibrium state where a perturbation is introduced and run on more than 1000 time steps. On the top of the figure, we show the resulting wing pressure distribution at $t = 3T$ (where T is the pseudo-period of the dynamics) and the wing geometry at initial time and $t = 3T$. Figure 13 shows a comparison of the generalized coordinates between MIMAS (dashed lines) and elsA's legacy reference solution (plain lines). The agreement between the original reference solution and the modular one is very good, although the coupling process is slightly different and the techniques for transferring fields, deforming the mesh are also different. Note that the signal is increasing with time due to the negative damping induced by the aerodynamic loads on the structure.

3.4. *Overheads and difference with elsA Ael module*

To demonstrate that our modular implementation does not add any overhead compared to the original implementation, we show in Table 1 the user time spent on a single node (24 cores) of ONERA's development cluster for elsA aeroelastic module (AEL) and MIMAS. Clearly for all cases, the overhead is not significantly enhanced and remains under the noise of the cluster node performance. We also show differences in terms of displacement and GAFs between the reference AEL module and MIMAS. For the static case, we observe 0.012 % of error in terms of maximum vertical displacement on the plane wing, while for the turbomachinery case, the difference in generalized aerodynamic force is below 1e-3 %. For the dynamic case, errors are larger due to the fact that the numerical temporal coupling scheme is different (see Section 2.6).

Table 1. User times and differences in quantities of interest compared to elsA aeroelastic module implementation.

	User time in elsA AEL (s)	**User time in MIMAS (s)**	**Difference in %**
Static coupling (CRM)	2.156+05	1.726e+05 s	0.012 (z-disp)
Harmonic forced (UHBR)	3.593e+05 s	3.634e+05 s	1e-3 (GAF)
Dynamic coupling (AGARD)	1.35e+04	1.424e+04 s	1e-1 (GAF)

4. Advanced aeroelastic applications with MIMAS

4.1. *The DLR-F25 plane case: CODA/NASTRAN coupling with HPC mesh deformation*

A first demonstration overcoming the conventional capabilities of elsA has been achieved with MIMAS on the DLR-F25 plane. This plane model is widely used as a benchmark in multi-disciplinary optimization and aeroelastic simulations. The goal of the calculation is to evaluate the static deformed shape of the plane during flight condition. The different improvements compared to previous standard calculation are the use of:

- an unstructured grid with tetrahedron, hexahedron and pyramids (see Section 2.2);
- a fast IDW method using clustering and layering (see Section 2.3);
- a compact RBF transfer method for displacements using the MUMPs library for solving the RBF system (see Section 2.4);
- a cleverer time-relaxation scheme using Aitken method to speed up convergence (see Section 2.6);
- a HPC CFD code of last generation (CODA) for the fluid step computation (see Section 2.8);
- a non-linear structural solver (here NASTRAN) directly exchanging data with the CFD code through the coupling tree of MIMAS (see Section 2.7).

Simulations at various angles of attack (from $-1°$ to $2°$) were conducted using both CODA and elsA within the MIMAS modular environment. An identical unstructured mesh of 10 million cells was used for both codes, and a similar numerical setup have been selected in order to be consistent for the comparison. The Roe solver is used in both codes, although in CODA, we employed the *spline quintic* flux limiter with a linear reconstruction, whereas for elsA we adopted a *minmod* limiter due to stability issue. For the structural solver, both linear and non-linear NASTRAN models have been considered, although the aim of this paragraph is not to show the differences between them (a complementary study would be more appropriate for that).

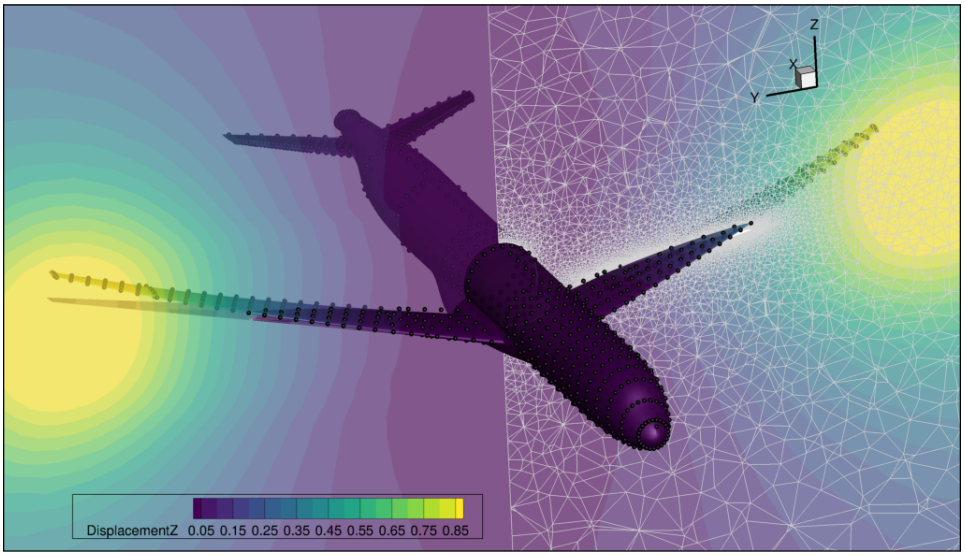


Figure 14. Mesh deformation of the DLR-F25 plane using IDW and clustering method (for angle of attack of 2°). Colours show the vertical displacement in the volume generated by the IDW algorithm on a 10-million-cells unstructured mesh composed of tetrahedrons, wedges, pyramids and hexahedrons.

Before running the aeroelastic calculations, tests have been performed to assess the performance and reliability of the new clustering IDW deformation method for unstructured meshes. Note that the elementary surfaces of the cells have been included in the IDW weights, to improve mesh quality. Rotation of cells are however not enabled here. Figure 14 shows the vertical deformation in the volume generated by the wall surface displacements. We checked that no negative cells are generated and that the mesh conserves a good quality during its deformation. Table 2 summarizes the performance of the method compared to the original brute force algorithm implemented in elsA aeroelastic module.

Table 2. User time for IDW mesh deformation algorithms on a single node (with 96 cores) of new generation ONERA's cluster. The first line corresponds to the time for preparation and computation during the first call of the algorithm. The second line is for the second coupling iteration until the end of the calculation.

	Brute force IDW (elsA)	**Clustered IDW (MIMAS)**
1st deformation including preparation	283 s	132 s
Next deformations	189 s	31 s

Clearly, using the clustering method for static calculations reduces computation costs to one-sixth of those required by the brute force algorithm, while achieving correct deformation. The preparation phase which includes the clustering procedure does not add any significant overhead. The mesh is deformed in almost 30 s at each coupling iteration. For comparison, fluid iteration steps in elsA last 225 s in average between each coupling. In CODA the time between these couplings varies strongly during the convergence history, and depends strongly on the

strategy used for convergence. The time spent on deforming the mesh remains thus around 10 % of the total time spent for the fluid resolution during each coupling. With old methods, this would have exceeded 100 % of the CFD time.

We further investigate elsA and CODA runs at an angle of attack of 2°. Different coupling strategies have been tested: the first one relies on a fixed coupling period and relaxation parameter equal to 0.75. Coupling is done after each 50 fluid iterations for CODA and 150 for elsA. In that case, the CODA run required in total 1300 fluid iterations using the GMRES method to reduce the density residual below 10^{-7}. For elsA, 5000 iterations with scalar LU relaxation were needed to reach a residual threshold of 10^{-5}. The second strategy relies on an Aitken under-relaxation scheme. By using this coupling strategy, no real improvement of the convergence has been achieved. The explanation might be that the convergence of the coupled system is primarily driven by the fluid and less affected by the structure because the wing flexibility remains low. Note that on contrary the convergence of the CRM plane (cf. Section 3.1) is strongly boosted by the Aitken under-relaxation scheme, probably because the flexibility is larger and the fluid Euler model converges intrinsically faster, so that the structure has more time to evolve during each coupling iteration.

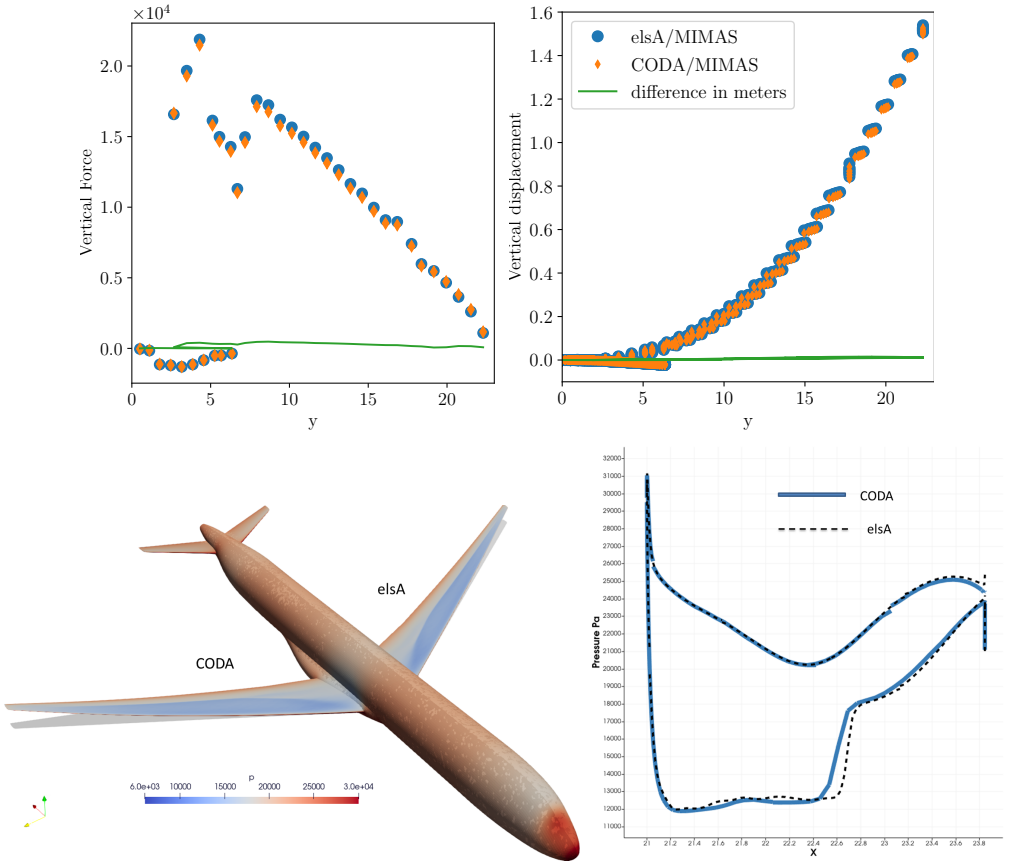


Figure 15. Comparison of the static simulation between elsA and CODA for the CFD part. Left top: Span distribution of vertical force on structural nodes. Right top: Distribution of vertical displacement on structural nodes. Left bottom: Pressure distribution on the plane with deformed shape. Right bottom: Pressure profile along the wall at the wing's half-span.

Figure 15 presents a comparison of the evolution along the span of the vertical displacement and force acting on the structural nodes when convergence is reached. There is a quite good agreement between the two runs, although the force amplitude close to the shock appears to be slightly smaller in CODA. The wing tip deflection is about 1.534 m for elsA and 1.52 m for CODA, which implies a 1.3 cm difference ($\leq 1\,\%$ of error). For further details on the simulations, we plot in the bottom panel of Figure 15 the pressure field on the plane for both runs (left side of the plane corresponds to CODA, right is elsA) and a comparison of the pressure profile along the wall at the wing's half-span. The shock intensity and position are relatively similar but the shock appears a bit smearer in the CODA run. This is possibly due to the fact that the order of reconstruction of the flux and the type of limiter employed in CODA differ from those used in elsA.

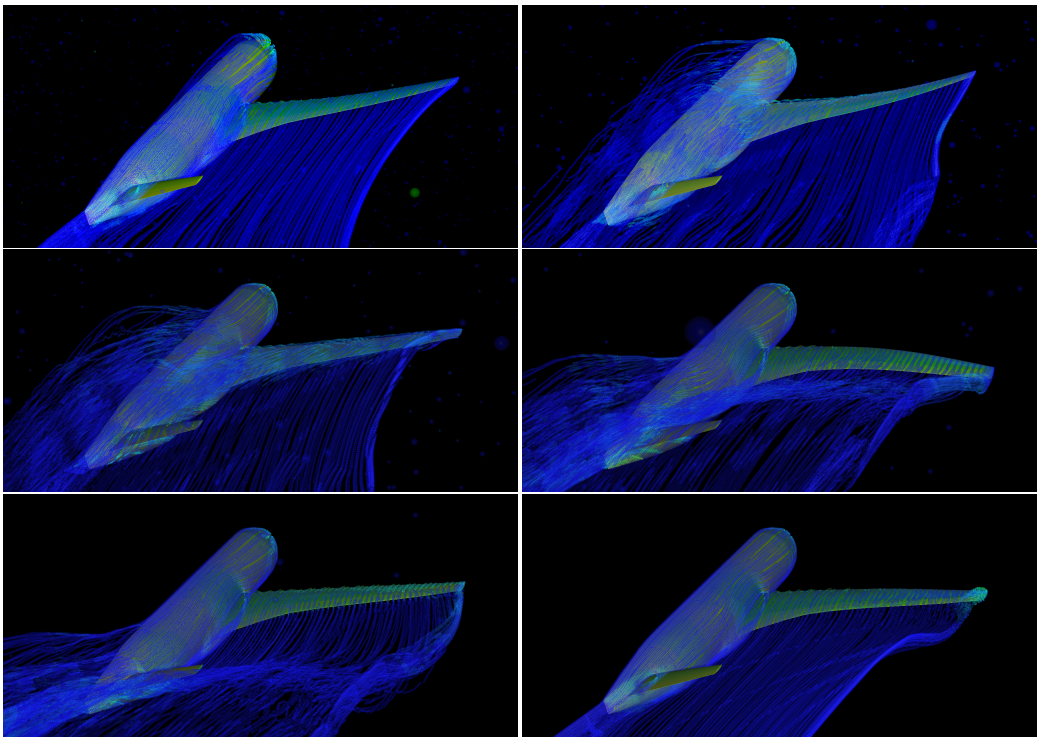


Figure 16. Flexible gust dynamic simulation of the CRM plane that couples the CFD code elsA and the CSM code NASTRAN in real-time, with mesh deformation at every time-step. Each code runs on a different machine, and data are transferred between them remotely using the Pyro Python module. The frames are respectively taken at 0.36, 0.63, 0.75, 0.92, 1.01 and 1.7 seconds after the launch of the gust. The blues lines represents the unsteady streamlines of the flow, computed in real-time during the calculation through a dedicated particle solver, independent of the CFD code.

4.2. *Dynamic aeroelastic response of the CRM aircraft subjected to gust loads*

Another new possibility with MIMAS is to couple directly the CFD code and the CSM code dynamically, while performing a mesh deformation at each time-step (or at regular dual steps of the cfd solver, see Section 2.6) to take into account the current structural displacements.

This is made possible with the new and faster mesh deformation algorithms, which absorb higher deformation. We show in this section a dynamical coupling achieved between elsA and NASTRAN, using the OpenFSI module of NASTRAN fully integrated within MIMAS. As both codes run on distant separate machines, we use the Python module Pyro to enable efficient in-memory data communication between solvers. The simulation presented here shows the response to a vertical negative wind gust characterized by a one-minus-cosine law, with an amplitude of 35 m and a wavelength of 50 m. The gust velocity is set to 100 m/s relative to the aircraft. The gust is imposed to the CFD code by adding to the mesh deformation a positive vertical grid velocity depending on time and space; this setup provides a rather strong gust that is largely above the certification test. However, our goal here is not to replicate actual aircraft tests, but rather to demonstrate MIMAS's capability to perform such coupling and to verify the robustness of the deformation algorithms when large displacements are involved. Both fluid and structural models are detailed in Section 3.1. For the NASTRAN model we restrict the simulation to a linear structure, but nothing prevents to run a nonlinear model, except the CPU time which can be significantly larger. Figure 16 shows a series of simulation snapshots, with the rendering enhanced by a particle tracer illustrating the flow streamlines (in blue). The effect of the gust is clearly visible on the third, fourth and fifth snapshots. The maximum displacement obtained on the wing tip is about 4 m (for comparison the static deformation at equilibrium at the wing reached 2.4 m) and −1.5 m at the gust maximum. To validate our direct coupling methodology, we compared it against a dynamic modal simulation, a coupling scenario which is already well-established and has been validated for the AGARD wing (see Section 3.3). In this reference case, the structural dynamics is restricted to a set of 10 modes, primarily representing the wing's bending and torsional behavior, computed in a pre-process phase. Note that for this reference run, the gravity and aerodynamic loads are directly projected onto the modal basis expressed on the fluid grid, while for the run in Figure 16, the forces and displacements are transferred at each time step using nearest neighbor algorithm and thin plate spline kernel. Figure 17 compares the two simulations in terms of lift coefficient and structural displacement at wing tip. The dynamic response is nearly identical, despite the fact that the underlying numerical algorithms used for coupling are fundamentally different.

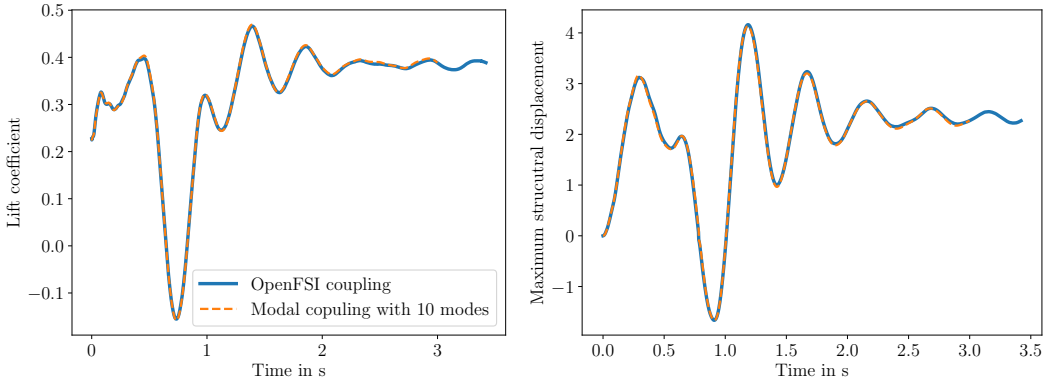


Figure 17. Comparison of the lift coefficient and wing-tip displacement between two gust aero-elastic simulations: one performed using a direct coupling between the CFD and CSM codes (with mesh deformation and data transfer at every time step), and the other using a modal coupling approach, similar to that presented in Section 3.3.

4.3. *The ERATO blade case: non-linear reduced order structural models coupled with CFD and VPM*

The final application presented in this paper focuses on the aero-acoustically optimized helicopter blade ERATO (Étude d'un Rotor Aéroacoustique Technologiquement Optimisé). The ERATO geometry was designed primarily by ONERA and the DLR in the '90s. Experimental studies were carried out in 1998 to aerodynamically characterize the blade for different advance ratio conditions and hover in ONERA's wind tunnel in Modane. The blade has a double sweep configuration to mitigate the effects of compressibility. Traditionally, blade structures are modelled by 1D approaches (rigid elements with stiff connections, FE beam formulation, etc.) and different studies have proven that for non-disruptive panforms the 1D structural modelling is accurate enough to represent the aeroelastic behaviour [55]. However, for ERATO like complex planform, even if the overall beahviour of the blade is captured, detailed studies highlighted the need for more advanced structural models [56] to correctly assess its aeroelastic behaviour. Recently, fully-coupled aeroelastic study with refined Reduced Order Models (ROMs) of the blade based on FEM in hover flight has been achieved with MIMAS [57].

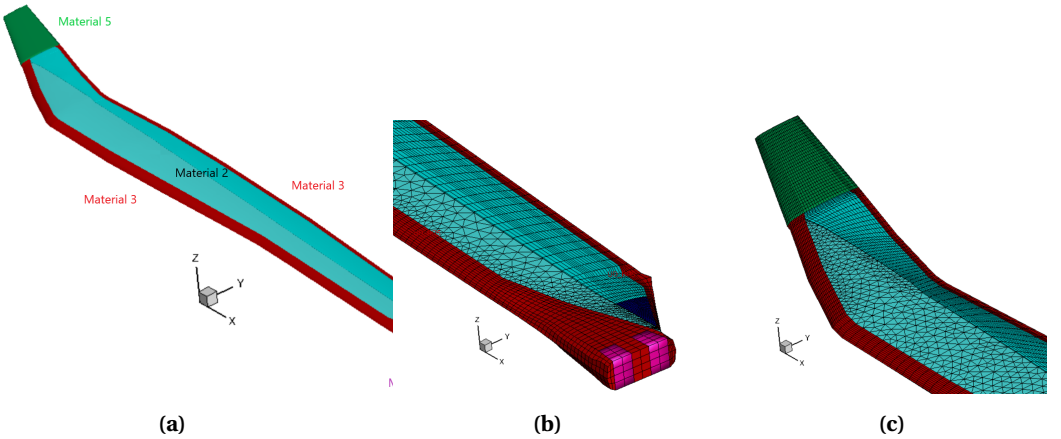


Figure 18. (a) Structural model of the ERATO blade used in this study. (b) Root mesh discretization. (c) Tip mesh discretization.

A detailed view of the root and the tip of the ERATO's structural mesh is presented in Figure 18. The mesh is formed by linear hexahedral, tetrahedral and pyramid FE elements and contains 32 436 nodes. The blade is composed of five different materials (represented by different colors) and is assumed to be clamped at the root. The blade structure is resolved through a geometrically nonlinear reduced model (ROMs) based on ICE projection (see Section 2.7). The construction of the non-linear model is carried out with the finite element software *Code_Aster* during a preprocess stage. To solve the fluid around the blade, several methods including URANS and mid-fidelity methods have been employed, with the aim to compare them. The most expensive computation is obviously the one performed with the elsA code using an overset technique. In that case, a Gear time algorithm is used with a Jameson, Schmidt and Turkel JST centred spatial discretization scheme (including artificial viscosity) and a Wilcox turbulence model for the hover flight. A less expensive method is the Vortex Particle Method (VPM) which is described in Section 2.9.

We summarize below some of the results conducted in this study. Figure 19 shows a snapshot of the VPM simulation during the coupling process. Approximately 30 new particles are launched

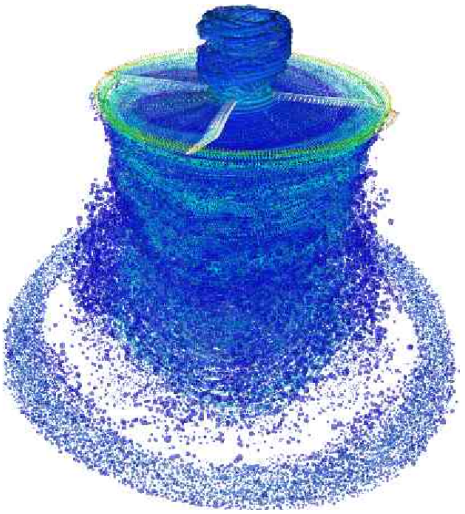
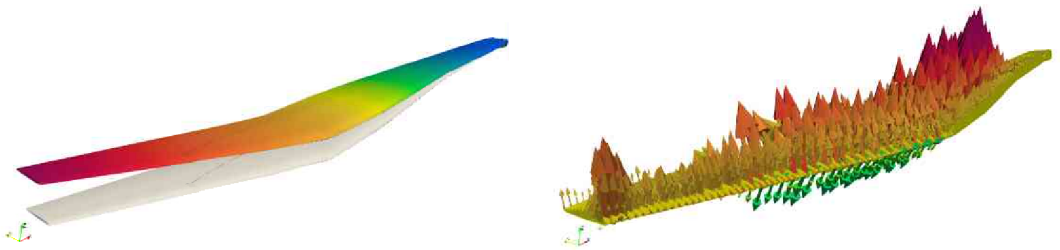


Figure 19. Snapshot of the wake generated by the VPM ONERA's code around ERATO's helicopter blades. The flow is represented by particles shed in the wake during the coupling process. The code has been run on a single GPU of the ONERA's development cluster.



(a) **(b)**

Figure 20. (a) Illustration of displacement transferred on the ERATO blade with the RBF algorithm, resulting from the non-linear reduced structural model. (b) Illustration of forces transferred on the ERATO blade with the nearest neighbour algorithm, resulting from the VPM computed forces.

from the chord quarter at relatively regular times, depending on the spatial and time variations of circulation. As explained in Section 2.9, the velocities of particles simulated in the wake are feeding back a lifting line model, which in turns change the particles distribution. CFD/ROMs and VPM/ROMs analyses show similar trends in terms of provided thrust, torsional deformation (pitch-down), flap displacements, aerodynamic loadings and maximum displacements, which suggest that mid-fidelity aerodynamic method can be employed to assess the aeroelastic behaviour of complex helicopter blades. Figure 20(a) shows that the ERATO blade in hover flight is significantly impacted by the torsional aeroelastic behaviour leading to a pitch-down effect on the blade-tip. In particular, simulations seem to indicate that the blade-vortex interaction is reduced when the flexibility of the blade is fully taken into account. Figure 20(b) illustrates the distribution of vertical loads on the structural nodes during the coupling process. Note that non-linear mechanical effects tend to slightly reduce the final blade deformation, although a more exhaustive study is required to characterise in details the non-linear physics.

5. Perspectives & Conclusions

This paper presents MIMAS, a new tool developed at ONERA designed to perform next-generation of aeroelastic simulations. MIMAS has been developed alongside the new emerging HPC codes such as CODA and SoNICS, with the aim of externalizing various simulation components that were traditionally embedded within elsA's fluid solver kernel. Among these components are the mesh deformation and data transfer algorithms, which have been greatly improved and parallelized to fit with the current HPC standard. Results were presented to demonstrate the reliability of this modular implementation across a wide range of test cases with multiple levels of complexity, including static coupling, forced harmonic motion, and dynamic coupling simulations. This modular framework also currently offers new capabilities, such as compatibility with unstructured meshes, which are essential for next-generation codes like CODA and SoNICS. Additionally, it offers several ways for coupling fluid with non-linear structures, and proposing new implementation of time-marching algorithm dedicated to FSI problems. The library supports now full dynamic coupling with linear or nonlinear structural models and codes (such as NASTRAN), extending its ability to simulate complex unsteady phenomenon with large deformations.

Future development within this library includes several key research areas aimed at enhancing its capabilities and performance. Firstly, efforts will focus on the linearization and adjoint formulation of all functions (or operators), allowing for more efficient optimization and sensitivity analysis. In addition, performance improvements will continue, particularly in the areas of mesh deformation and data transfer, which are critical for handling increasingly complex configurations. For elastic analogy in mesh deformation, an unstructured linear FEM code is under development and could progressively replace the structured version, with better preconditioners to accelerate its convergence. Interpolation methods will also continue to be refined, with a focus on blending techniques that take into account real intersections to improve accuracy. These techniques could also be upgraded to incorporate projection method based on 3D finite elements. Another important direction of research is the improvement of time-coupling schemes, which will allow more accurate and efficient simulations of dynamic systems. As the library evolves, it will become fully compatible with new HPC architectures such as those encountered in SoNICS and for example extend its capability to GPU execution. Extensions to other physics are also not excluded. Current developments, together with future improvements, will hopefully push the boundaries of what is possible in high-fidelity aeroelastic simulations.

Acknowledgments

The DLR-F25 is a conceptual design of an advanced short-to-medium-range airliner provided by DLR. It features an initial shape of a very high-aspect-ratio wing and was developed as a challenge for high-fidelity disciplinary and multi-disciplinary preliminary design. The development was funded by the German Federal Ministry for Economic Affairs and Energy as part of the LuFo project VIRENFREI.

Appendix A. Errors generated by clustering in the IDW formula

The purpose of this appendix is to determine the error bound made on the IDW formula when summing over the inverse distance of clusters barycentre.

Consider a set S of source points M_i and a set C consisting of N_c clusters of these points (see Figure 21). We assume that each point M_i belongs to a unique cluster. Each cluster is denoted by C_k, where k is an integer between 1 and N_c, and its cardinal is denoted n_k.

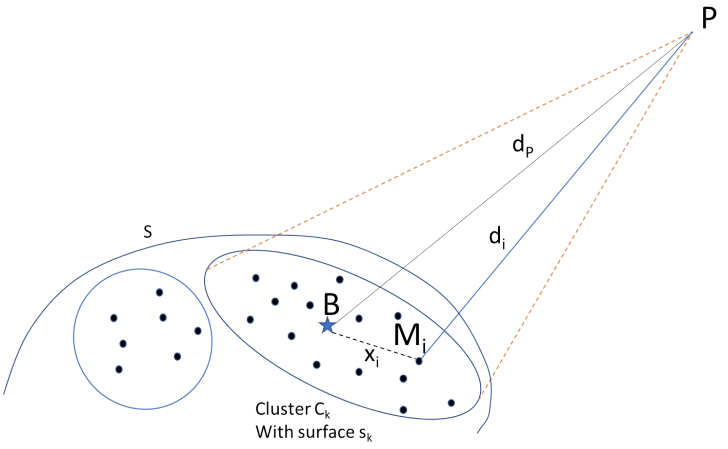


Figure 21. IDW clustering approximation.

For any summation over S, we can rearrange the terms as follows:

$$\sum_{i\in S} w_i = \sum_{k=1}^{N_c}\left(\sum_{j\in C_k} w_j\right). \tag{11}$$

We note with a bar and index k the average of a field over a cluster, then we have for any field w_i:

$$\sum_{i\in S} w_i = \sum_{k=1}^{N_c} n_k \overline{w}_k. \tag{12}$$

For a given cluster C_k, we introduce the dimensionless field dispersion ϵ_w with respect to the cluster barycentre (or average defined above) so that for each point in the cluster we have:

$$w_i = \overline{w}_k(1+\epsilon_{wi}). \tag{13}$$

It comes immediately that:

$$\sum_{j\in C_k} \epsilon_{wj} = 0 \qquad \text{and} \qquad \sum_{i\in S} \epsilon_{wi} = 0. \tag{14}$$

By using the above relations, the product of two fields w_i and D_i in the summation over the cluster leads to an additional correction term which depends on the product of the two dispersion fields:

$$\sum_{i\in S} w_i D_i = \sum_{k=1}^{N_c} n_k \overline{w}_k \overline{D}_k\left(1+\sum_{j\in C_k} \epsilon_{wj}\epsilon_{Dj}\right). \tag{15}$$

Now dividing (15) by (12) gives:

$$\frac{\sum_{i\in S} w_i D_i}{\sum_{i\in S} w_i} = \frac{\sum_{k=1}^{N_c} n_k \overline{w}_k \overline{D}_k\left(1+\sum_{j\in C_k} \epsilon_{wj}\epsilon_{Dj}\right)}{\sum_{k=1}^{N_c} n_k \overline{w}_k}. \tag{16}$$

We recognize on the left the exact IDW formula. On the right the approximated IDW formula computed from cluster barycentre is recovered for zero dispersion ($\epsilon_{wj} = \epsilon_{Dj} = 0$). The relative error $\mathscr{E}$ on the formula is then bounded by:

$$\mathscr{E}_{\max} = \max_{k=1,\dots,N_c}\left|\sum_{j\in C_k} \epsilon_{wj}\epsilon_{Dj}\right|. \tag{17}$$

The next step would then be to give an estimation of the upper bounds for ϵ_w and ϵ_D. However, in practice, this expansion cannot be used straightforwardly since the average of the IDW weight $\overline{w}_k$

cannot be computed for each point P in the volume without high computational cost. We search therefore for an approximation of the IDW formula that depends only on the distance from P to the clusters barycentre.

For an arbitrary point in space P, the IDW weight is:

$$w_j = \frac{s_j}{d_j^\alpha} \tag{18}$$

where

$$d_j^2 = \bar{\mathbf{x}}_P^2 + \mathbf{x}_j^2 - 2\mathbf{x}_P \cdot \mathbf{x}_j \tag{19}$$

is the distance from P to the point M_i in the cluster, $\bar{\mathbf{x}}_P$ is the position vector of P with respect to the cluster barycentre and s_i is an additional surface element relative to the source. In the following we note $d_P = \sqrt{\bar{\mathbf{x}}_P^2}$ the distance of P from the barycentre. Assuming that the squared distance inside the cluster $\mathbf{x}_j^2$ is small compared to d_P^2, we apply a Taylor expansion to compute the inverse distance:

$$\begin{aligned}\frac{1}{d_j^\alpha} &= \frac{1}{d_P^\alpha}\left(1 + \frac{x_j^2}{d_P^2} - 2\frac{\mathbf{x_j}\cdot\mathbf{x}_P}{d_P^2}\right)^{-\alpha/2} \\ &= \frac{1}{d_P^\alpha}\left(1 + \alpha\,\mathbf{x_j}\cdot\frac{\mathbf{x}_P}{d_P^2} - \frac{\alpha}{2}\frac{\mathbf{x}_j^2}{d_P^2} + \frac{1}{2}\alpha(\alpha+2)\left(\frac{\mathbf{x}_j\cdot\mathbf{x}_P}{d_P^2}\right)^2\right).\end{aligned} \tag{20}$$

Summing this expression over the points in the cluster C_k and assuming that the elementary surface into the sum is almost uniform gives:

$$\begin{aligned}\overline{w}_k &= \frac{1}{n_k}\sum_{j\in C_k}\frac{s_j}{d_j^\alpha} \\ &= \frac{1}{n_k}\sum_{j\in C_k}\frac{s_j}{d_P^\alpha}\left(1 + \alpha\,\mathbf{x_j}\cdot\frac{\mathbf{x}_P}{d_P^2} - \frac{\alpha}{2}\frac{\mathbf{x}_j^2}{d_P^2} + \frac{1}{2}\alpha(\alpha+2)\left(\frac{\mathbf{x}_j\cdot\mathbf{x}_P}{d_P^2}\right)^2\right) \\ &= \frac{s_k}{d_P^\alpha}\left(1 + \frac{\alpha}{n_k}\left(\sum_{j\in C_k}\mathbf{x_j}\right)\cdot\frac{\mathbf{x}_P}{d_P^2 n_k} - \frac{\alpha}{2d_P^2}\sum_{j\in C_k}\mathbf{x}_j^2 + \frac{1}{2n_k}\alpha(\alpha+2)\sum_{j\in C_k}\left(\frac{\mathbf{x}_j\cdot\mathbf{x}_P}{d_P^2}\right)^2\right).\end{aligned} \tag{21}$$

Since the sum of dispersion vectors is zero, we end up with

$$\overline{w}_k = \frac{s_k}{d_P^\alpha}\left(1 + \frac{1}{2}\alpha(\alpha+2)\frac{1}{n_k}\sum_{j\in C_k}\left(\frac{\mathbf{x}_j\cdot\mathbf{x}_P}{d_P^2}\right)^2 - \frac{\alpha}{2d_P^2}\frac{1}{n_k}\sum_{j\in C_k}\mathbf{x}_j^2\right). \tag{22}$$

Assuming that $\mathbf{x}_j$ and $\mathbf{x}_P$ make an angle δ_j, we finally obtain:

$$\overline{w}_k = \frac{s_k}{d_P^\alpha}(1+\epsilon_k), \tag{23}$$

with

$$\epsilon_k = \frac{\alpha}{2d_P^2}\frac{1}{n_k}\sum_{j\in C_k}\left[(\alpha+2)\cos^2(\delta_j) - 1\right]\mathbf{x}_j^2. \tag{24}$$

Since $0 < \left|\cos^2(\delta_j)\right| < 1$, an upper bound for ϵ_k is

$$|\epsilon_k| < \frac{\alpha(\alpha+1)}{2d_P^2}\frac{1}{n_k}\sum_{j\in C_k}\mathbf{x}_j^2. \tag{25}$$

We now use the fact that the clustering procedure ensures that the variance associated with coordinates and displacements inside each cluster is lower than a certain value σ_k. In other

words, if we note $\mathbf{x}_j$ the position of a point M_i in the cluster relative to the barycentre, and $\mathbf{D}_j$ the displacement field, we have:

$$\frac{1}{n_k}\sum_{j\in C_k}\mathbf{x}_j^2 \leq \sigma_k, \tag{26}$$

$$\frac{1}{n_k}\sum_{j\in C_k}\left(\bar{\mathbf{D}}_k-\mathbf{D}_j\right)^2 \leq \sigma_k. \tag{27}$$

Using these constraints gives:

$$|\epsilon_k| < \frac{\alpha(\alpha+1)\sigma_k}{2d_W^2} \tag{28}$$

where d_W is the minimal distance from P to the source, which can be easily computed in a pre-process calculation. Finally, going back to eq. 16, we have

$$\frac{\sum_{i\in S} w_i D_i}{\sum_{i\in S} w_i} = \frac{\sum_{k=1}^{N_c} n_k \frac{\bar{s}_k}{d_{P_k}^\alpha}\overline{D}_k\left(1+\sum_{j\in C_k}\epsilon_{wj}\epsilon_{Dj}\right)(1+\epsilon_k)}{\sum_{k=1}^{N_c} n_k \frac{\bar{s}_k}{d_{P_k}^\alpha}} \tag{29}$$

which leads to the following upper bound for the error

$$\mathscr{E}_{\max} = \max_{k=1,\dots,N_c}\left|\sum_{j\in C_k}\epsilon_{wj}\epsilon_{Dj} + \frac{\alpha(\alpha+1)\sigma_k}{2d_W^2}\right|. \tag{30}$$

Neglecting the variance of the displacement inside each cluster or assuming that the average cross correlation between displacements and IDW weight is negligible, we end up with:

$$\mathscr{E}_{\max}(P) = \frac{\alpha(\alpha+1)\sigma_{\max}}{2d_W^2}. \tag{31}$$

Declaration of interests

The authors do not work for, advise, own shares in, or receive funds from any organization that could benefit from this article, and have declared no affiliations other than their research organizations.

References

[1] V. de Gaudemaris, J.-S. Schotté, A. Placzek, L. Blanc and F. Thouverez, "Unsteady aerodynamic modeling of whirl flutter on a bending wing", *J. Phys. Conf. Ser.* **2647** (2024), no. 11, article no. 112013 (12 pages).

[2] V. de Gaudemaris, J.-S. Schotté, A. Placzek, L. Blanc and F. Thouverez, "Influence of aerodynamic modeling on the whirl flutter stability of a propeller under axial and non-axial flow conditions", 2024. Online at https://conf.ifasd2024.nl/proceedings/documents/150.pdf. Conference paper: International Forum on Aeroelasticity and Structural Dynamics (IFASD 2024).

[3] C. Koch and B. Koert, "Including Blade Elasticity into Frequency-Domain Propeller Whirl Flutter Analysis", *J. Aircraft* **61** (2024), no. 3, pp. 774–784.

[4] A. C. Gray and J. R. R. A. Martins, "A Proposed Benchmark Model for Practical Aeroelastic Optimization of Aircraft Wings", in *AIAA SCITECH 2024 Forum*, American Institute of Aeronautics and Astronautics, 2024.

[5] D. Gueyffier, S. Plot and M. Soismier, "SoNICS: a new generation CFD software for satisfying industrial users needs", 2022. Online at https://hal.science/hal-04045165/document. Conference paper: OTAN/STO/Workshop AVT-366.

[6] L. Cambier, S. Heib and S. Plot, "The Onera *elsA* CFD software: input from research and feedback from industry", *Mech. Ind.* **14** (2013), no. 3, pp. 159–174.

[7] S. Görtz, T. Leicht, V. Couaillier, M. Méheut, P. Larrieu and S. Champagneux, "CODA: A European Perspective for a Next-Generation CFD, Analysis and Design Platform", 2022. Conference paper: NATO AVT-366 Workshop on Use of Computational Fluid Dynamics for Design and Analysis: Bridging the Gap Between Industry and Developers.

[8] A. Gopinath and A. Jameson, "Time Spectral Method for Periodic Unsteady Computations over Two- and Three-Dimensional Bodies", in *43rd AIAA Aerospace Sciences Meeting and Exhibit*, American Institute of Aeronautics and Astronautics, 2005.

[9] N. L. Mundis and D. J. Mavriplis, "Toward an optimal solver for time-spectral fluid-dynamic and aeroelastic solutions on unstructured meshes", *J. Comput. Phys.* **345** (2017), pp. 132–161.

[10] C. Blondeau and C. Liauzun, "A modular implementation of the time spectral method for aeroelastic analysis and optimization on structured meshes", 2019. Online at https://hal.science/hal-02183133/document. Conference paper: International Forum on Aeroelasticity and Structural Dynamics (IFASD 2019).

[11] C. Liauzun and C. Blondeau, "A modular TSM solver for aeroelastic analysis and optimization", 2024. Online at https://conf.ifasd2024.nl/proceedings/documents/34.pdf. Conference paper: International Forum on Aeroelasticity and Structural Dynamics (IFASD 2024).

[12] G. Chourdakis, K. Davis, B. Rodenberg, et al., "preCICE v2: A sustainable and user-friendly coupling library", *Open Res. Eur.* **2** (2022), no. 51, pp. 1–47.

[13] M. Jadoui, *Solveurs de Krylov robustes pour la résolution partitionnée et monolithique du système adjoint couplé aéro-structure*, PhD thesis, Sorbonne Université (France), 2023. Online at https://theses.hal.science/tel-04336964v1/document.

[14] P. Girodroux-Lavigne, "Progress in steady/unsteady fluid-structure coupling with Navier–Stokes equations", *ONERA: Tire a Part* **1** (2005).

[15] 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 J.* **54** (2016), no. 3, pp. 828–846.

[16] B. Hallissy and C. E. Cesnik, "High-fidelity Aeroelastic Analysis of Very Flexible Aircraft", in *52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, American Institute of Aeronautics and Astronautics, 2011.

[17] T. A. Guimarães, C. E. Cesnik and I. V. Kolmanovsky, "An Integrated Low-Speed Aeroelastic-Flight-Dynamics Framework for Modeling Supersonic Aircraft", in *AIAA SCITECH 2022 Forum*, American Institute of Aeronautics and Astronautics, 2022.

[18] A. Dugeai and P. Vuillemin, "Highly flexible aircraft flight dynamics simulation using CFD", 2024. Online at https://hal.science/hal-04645957/document. Conference paper: International Forum on Aeroelasticity and Structural Dynamics (IFASD 2024).

[19] W. Liu, A. Skillen and C. Moulinec, *ParaSiF_CF: A Partitioned Fluid-Structure Interaction Framework for Exascale*, techreport, 2022.

[20] J. S. Gray, J. T. Hwang, J. R. R. A. Martins, K. T. Moore and B. A. Naylor, "OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization", *Struct. Multidiscip. Optim.* **59** (2019), no. 4, pp. 1075–1104.

[21] K. Boopathy and G. J. Kennedy, "Parallel Finite Element Framework for Rotorcraft Multibody Dynamics and Discrete Adjoint Sensitivities", *AIAA J.* **57** (2019), no. 8, pp. 3159–3172.

[22] J. R. Levesque, "The Code Aster: a product for mechanical engineers", *Epure* **60** (1998), pp. 7–20.

[23] E. Luke, E. Collins and E. Blades, "A fast mesh deformation method using explicit interpolation", *J. Comput. Phys.* **231** (2012), no. 2, pp. 586–601.

[24] N. Barral, E. Luke and F. Alauzet, "Two Mesh Deformation Methods Coupled with a Changing-connectivity Moving Mesh Method for CFD Applications", *Procedia Eng.* **82** (2014), pp. 213–227.

[25] N. R. Secco, G. K. W. Kenway, P. He, C. Mader and J. R. R. A. Martins, "Efficient Mesh Generation and Deformation for Aerodynamic Shape Optimization", *AIAA J.* **59** (2021), no. 4, pp. 1151–1168.

[26] P. Coulier and E. Darve, "Efficient mesh deformation based on radial basis function interpolation by means of the inverse fast multipole method", *Comput. Methods Appl. Mech. Eng.* **308** (2016), pp. 286–309.

[27] R. P. Dwight, "Robust Mesh Deformation using the Linear Elasticity Equations", in *Computational Fluid Dynamics 2006* (H. Deconinck and E. Dick, eds.), Springer, 2009, pp. 401–406.

[28] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data", in *ACM '68: Proceedings of the 1968 23rd ACM national conference* (R. B. Blue and A. M. Rosenberg, eds.), ACM Press, 1968, pp. 517–524.

[29] P. A. Burrough, R. A. McDonnell and C. D. Lloyd, *Principles of Geographical Information Systems*, Oxford University Press, 2015.

[30] W. J. Gordon and C. A. Hall, "Construction of curvilinear co-ordinate systems and applications to mesh generation", *Int. J. Numer. Methods Eng.* **7** (1973), no. 4, pp. 461–477.

[31] L. E. Eriksson, "Generation of boundary-conforming grids around wing-body configurations using transfinite interpolation", *AIAA J.* **20** (1982), no. 10, pp. 1313–1320.

[32] S. Sen, G. De Nayer and M. Breuer, "A fast and robust hybrid method for block-structured mesh deformation with emphasis on FSI-LES applications", *Int. J. Numer. Methods Eng.* **111** (2017), no. 3, pp. 273–300.

[33] J. Batina, "Unsteady Euler airfoil solutions using unstructured dynamic meshes", in *27th Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 1990.

[34] A. Dugeai, elsA: *Ael move3D Mesh Deformation Theoretical/User Guide*, techreport, ONERA, no. /ELSA/MU-10004-2014, 2014.

[35] D. Maruyama, D. Bailly and G. Carrier, "High-Quality Mesh Deformation Using Quaternions for Orthogonality Preservation", *AIAA J.* **52** (2014), no. 12, pp. 2712–2729.

[36] R. Franke and G. Nielson, "Smooth interpolation of large sets of scattered data", *Int. J. Numer. Methods Eng.* **15** (1980), no. 11, pp. 1691–1704.

[37] C. Farhat, M. Lesoinne and P. Le Tallec, "Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: Momentum and energy conservation, optimal discretization and application to aeroelasticity", *Comput. Methods Appl. Mech. Eng.* **157** (1998), no. 1, pp. 95–114.

[38] J. Samareh, "Discrete Data Transfer Technique for Fluid-Structure Interaction", in *18th AIAA Computational Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, 2007.

[39] A. De Boer, *Computational fluid-structure interaction: Spatial coupling, coupling shell and mesh deformation*, PhD thesis, Delft University of Technology (The Netherlands), 2008. Online at http://resolver.tudelft.nl/uuid:7e4cfc71-57eb-4a6e-9e90-38854de21ce2.

[40] T. C. S. Rendall and C. B. Allen, "Improved radial basis function fluid-structure coupling via efficient localized implementation", *Int. J. Numer. Methods Eng.* **78** (2009), no. 10, pp. 1188–1208.

[41] T. Achard, *Techniques de calcul de gradient aéro-structure haute-fidélité pour l'optimisation de voilures flexibles*, PhD thesis, CNAM (France), 2017. Online at https://theses.fr/2017CNAM1140.

[42] J. F. Kiviaho and G. J. Kennedy, "Efficient and Robust Load and Displacement Transfer Scheme Using Weighted Least Squares", *AIAA J.* **57** (2019), no. 5, pp. 2237–2243.

[43] G. E. Fasshauer, *Meshfree Approximation Methods with MATLAB*, Interdisciplinary Mathematical Sciences, World Scientific, 2007.

[44] N. Flyer, G. A. Barnett and L. J. Wicker, "Enhancing finite differences with radial basis functions: Experiments on the Navier–Stokes equations", *J. Comput. Phys.* **316** (2016), pp. 39–62.

[45] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent and T. Mary, "Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures", *ACM Trans. Math. Softw.* **45** (2019), no. 1, article no. 2 (26 pages).

[46] I. Ramière and T. Helfer, "Iterative residual-based vector methods to accelerate fixed point iterations", *Comput. Math. Appl.* **70** (2015), no. 9, pp. 2210–2226.

[47] A. Dugeai, Y. Mauffrey, A. Placzek and S. Verley, "Overview of the Aeroelastic Capabilities of the *elsA* Solver within the Context of Aeronautical Engines", *AerospaceLab J.* (2018), no. 14, article no. AL 14-03 (20 pages).

[48] J. J. Hollkamp and R. W. Gordon, "Reduced-order models for nonlinear response prediction: Implicit condensation and expansion", *J. Sound Vib.* **318** (2008), no. 4, pp. 1139–1153.

[49] T. Flament, J.-F. Deü, A. Placzek, M. Balmaseda and D.-M. Tran, "Reduced-order model of geometrically nonlinear flexible structures for fluid-structure interaction applications", *Int. J. Non-Linear Mech.* **158** (2024), article no. 104587.

[50] T. Flament, J.-F. Deü, A. Placzek, M. Balmaseda and D.-M. Tran, "Reduced Order Model of Nonlinear Structures for Turbomachinery Aeroelasticity", *J. Eng. Gas Turbines Power* **146** (2023), no. 3, article no. 031005 (9 pages).

[51] L. Hascoet and V. Pascual, "The Tapenade Automatic Differentiation tool: principles, model, and specification", *ACM Trans. Math. Softw.* **39** (2013), no. 3, article no. 20 (43 pages).

[52] G. S. Winckelmans and A. Leonard, "Contributions to Vortex Particle Methods for the Computation of Three-Dimensional Incompressible Unsteady Flows", *J. Comput. Phys.* **109** (1993), no. 2, pp. 247–273.

[53] J. Valentin, L. Bernardos, É. Rivoalen and G. Pinon, "Vortex Particle Velocity Diffusion Method Using Dynamic Turbulence Control Based on Enstrophy", 2024. Online at https://hal.science/hal-04703933.

[54] E. C. Yates, N. S. Land and J. T. Foughner, *Measured and Calculated Subsonic and Transonic Flutter Characteristics of a 45 sweptback Wing Planform in Air and in Freon-12 in the Langley Transonic Dynamics Tunnel*, NASA technical note, National Aeronautics and Space Administration, 1963.

[55] R. Moretti, H. Yeo, F. Richez and B. Ortun, "Rotor Loads Prediction on UH-60A Flight Test using Loose Fluid/Structure Coupling", 2023. Online at https://hal.science/hal-04115739/document. Conference paper: Vertical Flight Society 79th Annual Forum & Technology Display.

[56] M. Balmaseda, H. Yeo, B. Jayaraman and B. Ortun, "High-fidelity aerodynamic loads analysis of the double-swept ERATO rotor", 2024. Conference paper: 50th European Rotorcraft Forum (ERF 2024).

[57] M. Aguirre, A. Riols-Fonclare and F. Richez, "Aeroelastic Analysis with Rapid Methods of the Double-swept ERATO Blade with an Homogeneous Structure in Hover Flight", 2024. Conference paper: Vertical Flight Society 80th Annual Forum & Technology Display.

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus. Mécanique
2025, Vol. 353, p. 1261-1287
https://doi.org/10.5802/crmeca.334

Research article

SoNICS: design and workflow of a new CFD software

Michael Lienhardt [*, a] **and Bertrand Michel** [b]

[a] DTIS, ONERA, Université Paris-Saclay, 91120 Palaiseau, France

[b] DAAA, ONERA, 92320 Châtillon, France

E-mails: michael.lienhardt@onera.fr, bertrand.michel@onera.fr

Abstract. SoNICS is a new CFD software which builds upon the experience acquired in the years developing and extending the *elsA* CFD tool implemented at ONERA and used in industry. This article presents an overview of the SoNICS design and workflow, the motivation for this new tool, its structure and how it manages the user configuration (including the input mesh) to produce in the end a graph of tasks to schedule. In particular, we will discuss how well-studied concepts like variability, task graphs and compilations passes are cornerstones of the SoNICS workflow and allow for: a relatively easy evolution and maintenance of the code-base; a relatively straightforward user experience; and good performances.

Keywords. Computational fluid dynamics, variability, code assembly, compilation passes.

Note. Article submitted by invitation.

Manuscript received 19 January 2025, revised 9 October 2025, accepted 20 October 2025.

1. Introduction

elsA is an important Computational Fluid Dynamics (CFD) solver that was improved and extended over more than 25 years of development [1–3]. It was and still is developed in partnership with and used by several industrial partners such as Airbus, Safran, EDF or MBDA. *elsA* is now a tool with many capabilities, and can be used both for transport aircraft configurations [4,5] and helicopter applications [6,7], as well as for turbomachinery flow simulations [8,9] and steam turbine applications [10,11].

However, *elsA*'s implementation was not designed to tackle the challenges that were raised in the past decade. Indeed, it has a monolithic structure based on oriented object approach that combines python, C++ and Fortran code with intricate dependencies. This structure together with the large code base implementing the many functionalities supported by *elsA* makes it difficult to maintain and extend, which includes adding support for new efficient hardwares such as GPUs. Moreover, to support gradient optimization, the code differentiation of all kernels in *elsA* must be hand-written instead of automatically generated using either existing static tools [12,13] or dynamic analysis [14–16]. Updating *elsA* to support GPUs and automatic differentiation would require a complete overhaul of its implementation.

* Corresponding author

In addition to these difficulties, the usecases for fluid flow simulation evolved in the past decades, and new functionalities are expected by our industrial partners or by internal requests from ONERA's teams. In particular, we identified three main requests from *elsA*'s users.

Simulations: There is an important interest in simulating the flow around or inside larger objects and with more precision, including the simulation of full engines. Such simulations are complex to configure and require much memory and computational resources to perform. A modern CFD software must be easy to configure and must be able to take advantage of powerful hardware, that are all complex, distributed and heterogeneous.

User experience: Setting up a simulation, checking that it is going smoothly and analysing the results are arduous tasks, and a CFD tool should integrate tools and techniques that would make them as straightforward as possible. This includes simple mechanisms to help the user to design their mesh (such as automatic mesh refinement), and APIs to help couple the tool with other solvers. Many elements can contribute to a better user experience (e.g., better user interfaces, clear and interactive feedbacks), but all require a robust understanding of what is the information relevant to the user, what computation is being performed, and how to transform this computation to perform additional task, like exchanging data with another solver or computing mesh refinement metrics and updating the mesh.

Design: Simulations can be used to help design objects by pointing what features of a simulated artifact cause turbulences or other unwanted behaviors. This is done by differentiating the performed simulation, and so a CFD tool should provide a general mechanism to produce such derivatives so the end users can get all the data they need to analyse and optimize their artifact.

This paper presents the new CFD tool SoNICS which builds upon the expertise developed around *elsA*, and uses a new implementation design based on technics inspired from software engineering, scheduling, and other to solve the limitations of *elsA*'s implementation and answer our partners' requests.

This article is structured as follows: Section 2 presents the main choices made in the design of SoNICS and motivates how they contribute in satisfying the user requests; Section 3 briefly presents and discusses some of the simulation that were already done using SoNICS; Section 4 discusses the related work and Section 5 concludes the article.

2. SoNICS's design and workflow

As shown by the work around *elsA* and other CFD solvers, designing such a tool is a very difficult task. The main difficulty is clearly captured by the apparent contradiction raised by the **Simulations** requirement given in Section 1. Indeed, a CFD solver answering this request must be able to perform a large panel of computations on different hardware architectures, which requires to have a large configuration space (e.g., which turbulence model to apply, how many CPUs are available), and a complex control flow to apply this configuration during the computation. This control flow entails a large running overhead, which contradicts the requirement for the computation to be efficient. Moreover, compilers also have difficulties to manage codes with complex control flow, e.g., even if the field of code vectorization is the subject of much research [17–19], all the technics must have a complete knowledge of the loops to vectorize and fail with conditionals. Finally, automatic differentiation, required by the **Design** requirement, can be achieved on code with complex control flow with a well known technic [14–16] which unfortunately relies on registering at runtime all performed operations, which can create a large memory overhead as discussed in [20]. On the opposite, purely functional code can easily be differentiated using source transformation [12,13] with no overhead during execution.

A second difficulty is captured by another contradiction raised by the **Simulations** and **User experience** requirements given in Section 1. Indeed, on one hand, the **Simulations** requirement asks a CFD solver to be able to perform a large range of simulations, which implies providing to the user an extensive panel of ways to configure the solver, including many options and hooks into its code. On the other hand, the **User experience** requirement asks a CFD solver to provide every user with a clear and expressive interface dedicated to their needs, to help them as much as possible in setting up their simulation and pinpointing configuration errors before starting any computation.

The development of *elsA* faced these difficulties at a lesser level, and proposed an interesting solution for the main difficulty, that extracts its control flow from its computation. Indeed, the control flow of a CFD solver has three main objectives: (i) it selects which variant of a functionality to execute, depending on the user configuration (e.g., which turbulence model to apply); (ii) it distributes the computation over the hardware architecture and identifies the necessary communications between the nodes; and (iii) it triggers the computation of values required by a functionality, depending on which variant of the functionality is being executed and on the mesh topology. Since none of the inputs of the control flow change during the computation, the selection, distribution and triggers can be computed once and for all, and the computation can then be performed without any overhead. *elsA* implemented its control flow in a complex component called *Factory*, and the computation's implementation was split into many independent kernels performing a well-identified task (e.g., computing the gradient of a value), called *operators*, that the *Factory* would combine to produce a computation. This design has three additional advantages:

(1) operators are simpler to understand and implement than complete functionalities, which improves their capability to be checked, tested and maintained;
(2) many functionalities share some computations, and so splitting them into operators performing these computations avoids code duplication and also avoids possibly computing multiple time the same data at runtime;
(3) it is easier to update, extend and add new functionalities with such a modular implementation, since existing operators can be used in new functionalities, and new ones can be seamlessly integrated.

However, *elsA*'s *Factory* is difficult to maintain and extend, has difficulties to manage complex distribution schemes, and is completely unable to handle automatic differentiation.

SoNICS's design uses the same approach to keep *elsA*'s interesting properties, but completely replaces the *Factory* with an explicit workflow based on the three following technics:

(1) it uses a standard and well studied formalism to manage its many configuration options called *Software Product Lines* (SPLs) [21];
(2) it models the computation as an acyclic graph of operators, which enables SoNICS to perform a very large range of analysis and optimization that were not possible in *elsA*, including having a very efficient distribution and communication scheme;
(3) in particular, SoNICS currently implements four graph transformation algorithms that manage: (i) memory optimization and allocation; (ii) building the communication between the different hardware nodes; and (iii) performing the automatic differentiation.

The rest of the section is structured as follows: we first discuss how we structure SoNICS's user interface to answer the **User experience** requirement; we present the complete SoNICS workflow that replaces *elsA*'s *Factory*; and then we discuss in more details the different aspects of the acyclic graph of operators and its transformations.

2.1. *User interface*

Configuring a simulation is a difficult task, which involves setting up which physical model to use, which numerical scheme, which constants to use, and creating a mesh that could capture the phenomena of interest. Unfortunately, no solution was found to simplify the configuration process for a general purpose CFD tool. However, similar simulations are usually also configured similarly, and so it makes perfect sense to design a wrapper tool specific for a category of simulations that would only ask the user for the few information that would change in configuring a simulation of that category, and automatically extend it into a full configuration for the CFD tool.

To simplify the direct configuration of SoNICS and the design of wrapper tools, we identified from our experience with *elsA* the central elements in configuring a simulation, and designed a way to make these elements as natural to use as possible, and also focused on error reports, so the user, if their configuration is erroneous, could easily identify what the problem is and how to solve it.

SoNICS's user interface is constructed around four configuration elements.

Topological data: We use the CGNS (CFD General Notation System) standard [22,23] to store the topology and its associated data. This format is widely used for storing and sharing CFD data (mesh structure in particular) and provides a clear hierarchical organization of data, facilitating the management of complex mesh structures with a clear separation of geometry, topology and solution data. Moreover, ONERA has developed the *Maia* library [24] which facilitates the manipulation of CGNS data structures using a simple python API. *Maia* provides easy access to complex algorithms such as load/store operations, partitioning, and solution transferring within a distributed memory context. Finally, to enhance modularity and performance in distributed memory environments, ONERA has implemented a simple and efficient in-memory representation of the CGNS format. This approach allows for efficient data handling and manipulation, which is crucial for high-performance computing applications. The integration of the in-memory CGNS representation is effectively utilized in the SoNICS framework, supporting the entire workflow from pre-processing to post-processing.

Options: As discussed previously, we use SPL technics to manage the organise the many options of SoNICS, and in particular, these options are organised into a *Feature Model*. Feature Models have the interesting property of capturing the notion of *valid* option selections. Indeed, not all options can be selected together. This property allows SoNICS to detect and report possible errors in the user's option selection before starting its computation.

Data of interest: The user may be interested in data that are not part of the computation of the fluid flow, e.g., the pressure on a specific boundary to check if the material is resistant enough, or some metrics used for automatic mesh refinement. *elsA* uses an enumeration to list all data that can be requested by the user, which forbids the user to request data that could be computed but that was not included in the enumeration. In SoNICS, we designed a *Domain Specific Language* (DSL) embedded in python to express any possible data: this way, any data that can be computed by SoNICS can be requested by the user.

Workflow modification: Finally, some user requests have an impact on SoNICS's workflow. First, coupling with another solver requires to regularly exchange data with another software: to this end, SoNICS includes a very simple *Aspect Oriented Programming* [25–27] interface called *triggers* which allows the user to insert any python code before and after each iteration of the simulation. Moreover, automatic mesh refinement wraps the whole SoNICS's workflow in a loop where each iteration computes some refinement metrics and adapts the mesh w.r.t. these metrics, or exits the loop when the obtained mesh is satisfactory. To allow such an intrusion into SoNICS's workflow, it has been

implemented in python with clear modules and data structures, so any motivated user could insert their code where it is relevant.

2.2. *Complete workflow*

The complete workflow of SoNICS is presented in Figure 1. It produces from the user configuration a well-balanced distributed computation performing the requested simulation. The workflow starts from the left: we have the *Operator Bank*, filled by the SoNICS developers, which stores all the information concerning the operators available in SoNICS; and the user configuration split into the *Option Selection*, the *CGNS input file* and the *Other user requests*. The *Option Selection* is used by the *Variant Selection* mechanism to obtain the *Operator Variants* from the *Operator Bank*. This mechanism is a standard tool in the SPL formalism, which will be discussed in Section 2.3. In parallel to the *Variant Selection*, the raw mesh information, called *DistTree* is extracted from the CGNS file. Using the *DistTree*, the *Variant Selection* and the data of interest in the *Other user requests*, we generate a *Primal Operator Graph* whose purpose is to identify the computation that needs to be performed, and create a uniform distribution of that computation. This distribution is computed by the *Distribution* component, which is currently implemented using SCOTCH [28,29] or ParMETIS [30,31], and is used to produce a distributed mesh called *PartTree*. Using this distributed mesh, SoNICS produces a *Distributed Operator Graph*: this is the graph that will be used during SoNICS's computation. This distributed graph is then managed by the *Builder* which fetches the implementation of the different operators, gets from the *PartTree* the different constants needed by the operators, resolves the inputs and outputs of the operators into actual pointers, and produces a distributed *taskflow* that can be executed by the runtime. The runtime is thus responsible of performing the computation given in the input taskflow, and also performs the additional tasks requested by the user in the *Other user requests*. It is composed by many libraries: some, like *ParaDiGM* [32], are used to manage the mesh, the data hosted by it and the communications between the different computation nodes; some, like *taskflow* [33] are used to execute the graph both on CPU and on GPU; others, like the *array manager* are internal to SoNICS to perform various tasks (e.g., the *array manager* implements all of the data allocation and reallocation functionalities required to execute the graph's operators).

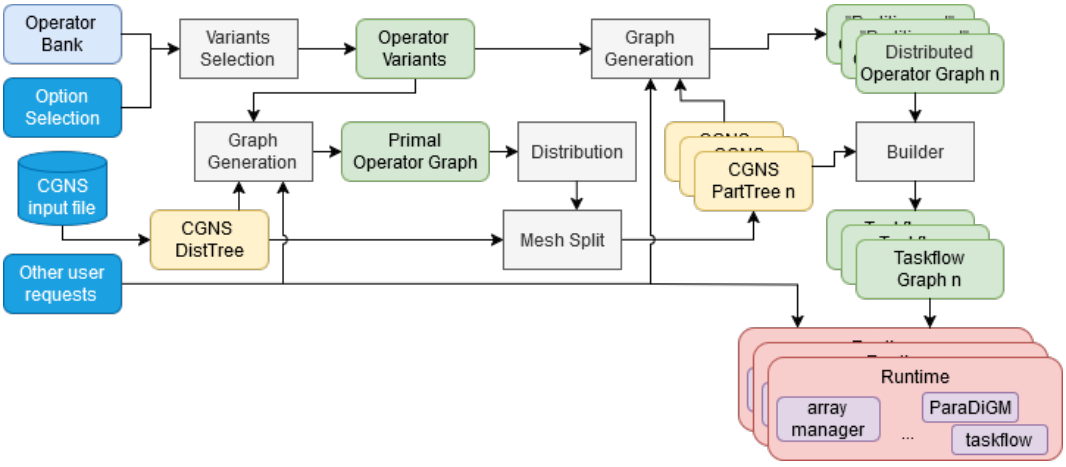


Figure 1. SoNICS's main workflow.

2.3. *Software product lines*

Conceptually, a Software Product Line (SPL) corresponds to a set of similar programs, called variants, that share many characteristics and only differ due to well identified feature selection [21]. For instance, the linux kernel is a well-known SPL [34]: while every running linux kernels do implement an operating system, depending on the selected and unselected options before compiling it, many of the functionalities of that system do change (e.g., hardware support, dynamic module loading, etc). SPLs are largely used in the automotive industry [35] to manage the software of the many car variant. The field of SPL focuses on solving two problems: (i) how to organize the many options of a software to have a clear picture of the configuration space; and (ii) how to apply these options in the code to avoid code duplication and facilitate maintenance and evolution.

A CFD solver such as SoNICS is a natural usecase for SPL, since it must include many options to capture all the possible user configurations. In SoNICS, we used a well-known standard to organize its options, called *Feature Model* (FM). Such a model organizes options in a tree hierarchy, with additional constraints on option selection written with boolean formula. A simplified version of SoNICS's FM is presented in Figure 2. Here, the configuration space of SoNICS (modeled with the `sonics` option) is split in four categories: `formulation` states which discretization of the computation to use (e.g., either computes the values in the cells of the mesh, or on its vertices); `spacial_dimension` states how many dimensions to consider (2 and 3 dimensions are currently supported); `motion` states if the computation is stationary or not; and `model` presents the options to configure the equations to consider in the computation. Below the tree hierarchy are two additional constraints restricting how the options can be selected: $\texttt{transition\_closure} \Rightarrow \texttt{kl} \vee \texttt{kw}$ means that if the user wants a transition closure, then only the turbent model `kl` or `kw` can be selected; and $\texttt{zdes} \Rightarrow \texttt{spalart}$ means that the Zonal Detached Eddy Simulation can only be performed in SoNICS with the spalart turbulence closure.

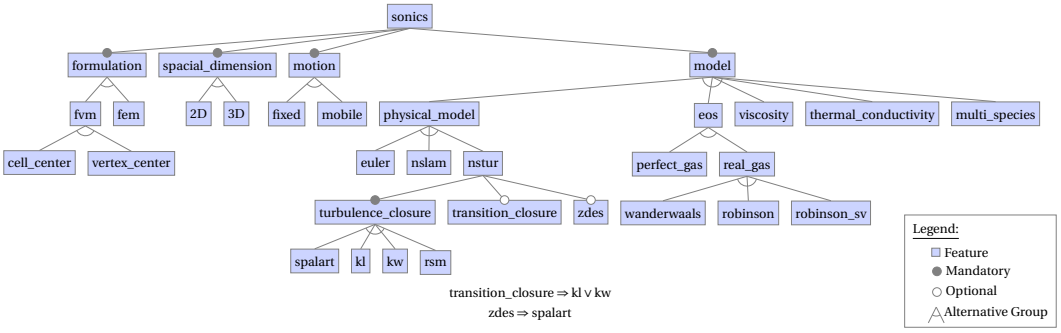


Figure 2. A simplified version of SoNICS's feature model.

The effect of most of these options is to select which variant of every operators implemented in SoNICS will be used in the simulation, and to generate some additional information that will be discussed later. This selection, performed by the *Variant Selection* in Figure 1, is implemented by a combination of *Delta-Oriented Programming* [36,37] which carries the options selection to the operators, and either brute force implementation of an operator (one implementation per variant), or code generation technics based on C++ templates or sympy [38,39].

2.4. *Task graph*

While the SPL mechanism manages the selection of the operators' variants, the operator graph captures which operators are necessary for a given computation. This graph structures the computation into two kinds of nodes, operator nodes and data nodes, and where the edges connect operators to their output data and data to the operators using them in input.

The *Graph Generation* task in SoNICS's workflow automatically produces this graph, and works by: first producing a *local operator graph* describing only the computation of the direct data; and then applying the different algorithms that will be discussed in Section 2.5 to complete the graph with derivative computation, memory allocations, etc. In this section, we present how the initial local operator graph is generated.

The first step of the *Graph Generation* is based on two important characteristics of many HPC computations: (i) the computation calculates well identified values (e.g., in SoNICS, these values are the flow update and other values explicitly requested by the user); (ii) the steps that perform this computation are mostly side-effect-free, i.e., they have well identified inputs and outputs and do not have complex interaction patterns using shared objects or global data. Using these two characteristics, the local operator graph is generated, using the operators stored in the *Operator Variants* registry with the following pseudo-algorithm:

```
1 todo = set of all values to be computed
2 done = ∅
3 while todo ≠ ∅:
4   v = remove a value from todo
5   f = find an operator that computes v
6   done = done ∪ {v}
7   todo = todo ∪ (inputs(f) \ done)
```

This algorithm starts with the values that need to be computed (i.e., flow update and the user's data of interests) and iteratively adds operators to compute these values, these operators having themselves inputs that need to be computed. Hence, it is essential for this algorithm to have a clear description of the inputs and outputs of every available operators: this description is one of the additional information generated during the *Variant Selection* and is expressed using the DSL also used by the user to specify their data of interest. Moreover, an important property of a CFD simulation is that its computation depends on the mesh structure: this is captured in SoNICS with operators having a generic number of inputs, that are instantiated with the mesh topology. Figure 3 introduces our running example, which consists of the generation of a very simple operator graph, considering a euler computation of order 1, including a source term, on a mesh with one zone with two border conditions, one of type *Inj* and one of type *wallslip*.

The generation starts with the value modeling the flow update, called `Rhs`, on the only zone of the mesh. One iteration of the loop finds the operator `Rhs` which computes `Rhs` and has the value `Balance` as input. After five other iterations, the operators `FluxBalance`, `SourceTerm1`, `FluxDensity`, `ConvectiveFlux` and `Primitive` were found and inserted in the graph. The next iteration adds the operator `ConvectiveFluxBC` to compute F^c_{BC}: as discussed before, the inputs of this operator depend on the structure of the mesh, and since the mesh's zone has two boundary conditions, this operator has two inputs, one per condition. The rest of the graph generation is straightforward: the two `Conservative` values on the boundary conditions are computed by their respective operators, both depending on the main `Conservative` data. The only value left without an operator computing it is `Conservative`, which is the input of the global computation.

To illustrate how option selection can affect the computation, Figure 4 presents the graph for the same computation, except that its order has been changed to 2. In order 2, the operator

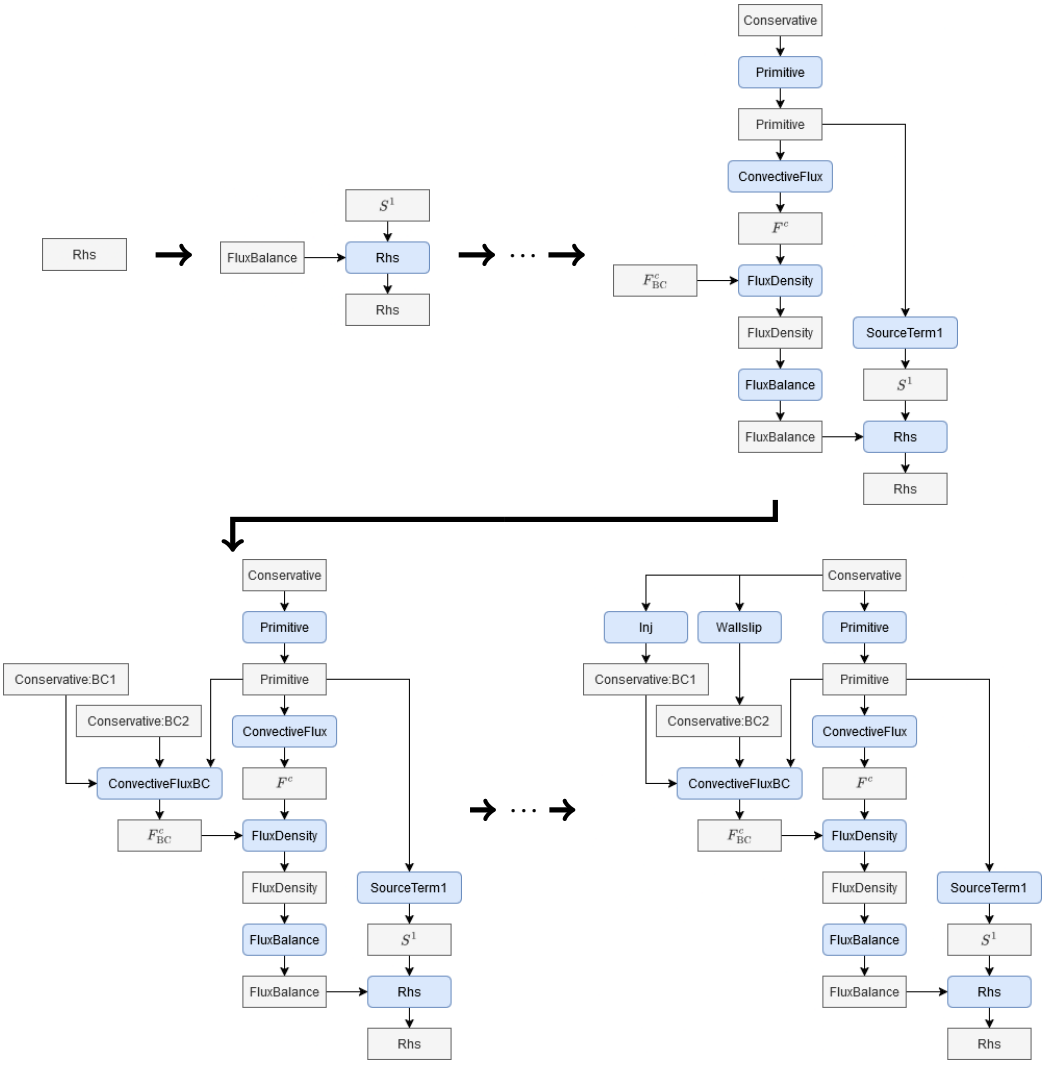


Figure 3. Simple operator graph generation.

`ConvectiveFlux` additionally needs the value `grad(Primitive)` in input. While the rest of the generated graph is identical to the one in Figure 3, the graph generation process includes this new dependency and finds the operator `Gradient` to compute it from the data `Primitive`.

One of the main challenges in creating this automatic graph generation mechanism is to design the DSL in which to express the inputs and outputs of the operators. Since the operators have various properties (e.g., some have a generic number of inputs; others like `Gradient` can produce the gradient of any value), that language must be flexible enough to capture these properties, while still allowing to easily find an operator computing a given value (line 5 of the graph generation algorithm). The details of the language we use in SoNICS are discussed in [40,41].

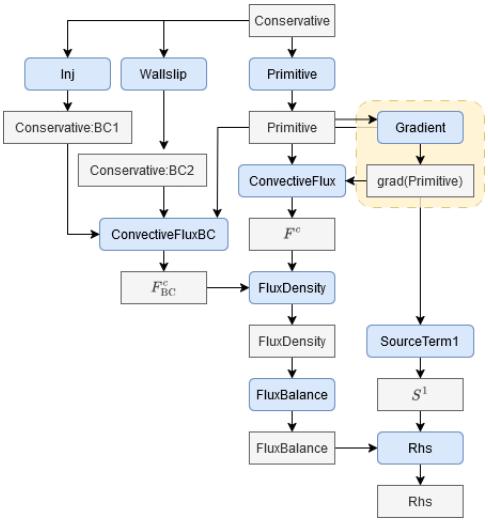


Figure 4. Simple operator graph with order 2.

2.5. *Graph transformation*

These different algorithms are implemented and managed in a similar manner as *passes* during a compilation process: they analyze the operator graph, extract relevant information from it, and modify it w.r.t. this information. The advantage of this architecture is its modularity: (i) every pass has one specific purpose and so like for operators, they are simpler to understand and implement than a complete transformation process, which improves their capability to be checked, tested and maintained; (ii) every pass is independent, and so it is easy to enable, disable or implement new passes depending on the needs. In SoNICS, we have four main passes.

2.5.1. *Distributed computation*

This pass manages the distributed nature of the computation by ensuring that all distant data are computed and adding *data transfer and communication* operations to the graph. Indeed, stencil operations use as input data computed on neighbouring zones: we thus need to ensure that this data is computed, and perform the transfer to make it available for the stencil operations. This pass thus analyses the graph to identify the stencil operators in the graph and the distant data they need, ensure that these data are computed and add the corresponding data transfer operators.

Let us illustrate this pass on the operator graph in Figure 3. The starting point of the pass is the operator graph with some *stencil information*. This stencil information is provided either by the developer (for hand-written operators) or can be automatically extracted from sympy code. Figure 5 presents the stencil information for the operators `Primitive`, `ConvectiveFlux` and `Rhs`. That information gives for every operator the *rank*, i.e., the number of layer of halo cells, required for each of its inputs, and the rank provided for each of its outputs. For instance, Figure 5 states that the `Primitive` operator, having the same rank for its input and output data, does not have a stencil and can be applied on data with a halo of arbitrary size. `ConvectiveFlux` is a stencil operator, since that even if it can be applied on data with a halo of arbitrary size, the halo produced in output is one less than the halo required in input. Finally, `Rhs`, like `Primitive` is not a stencil operation.

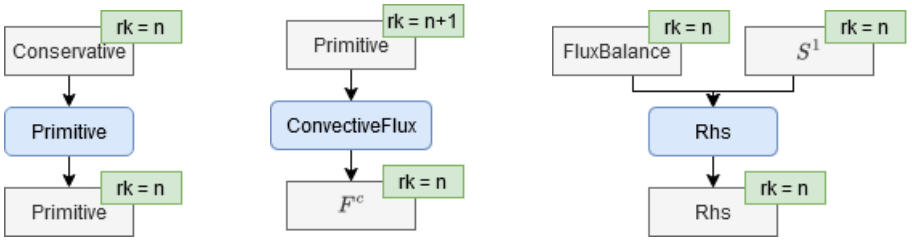
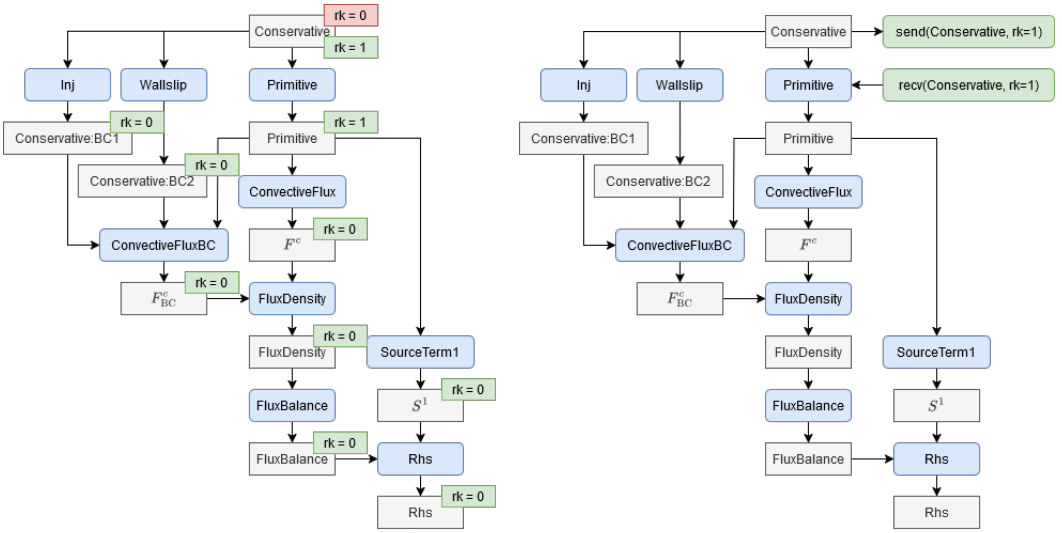


Figure 5. Stencil information of the operators `Primitive`, `ConvectiveFlux` and `Rhs`.



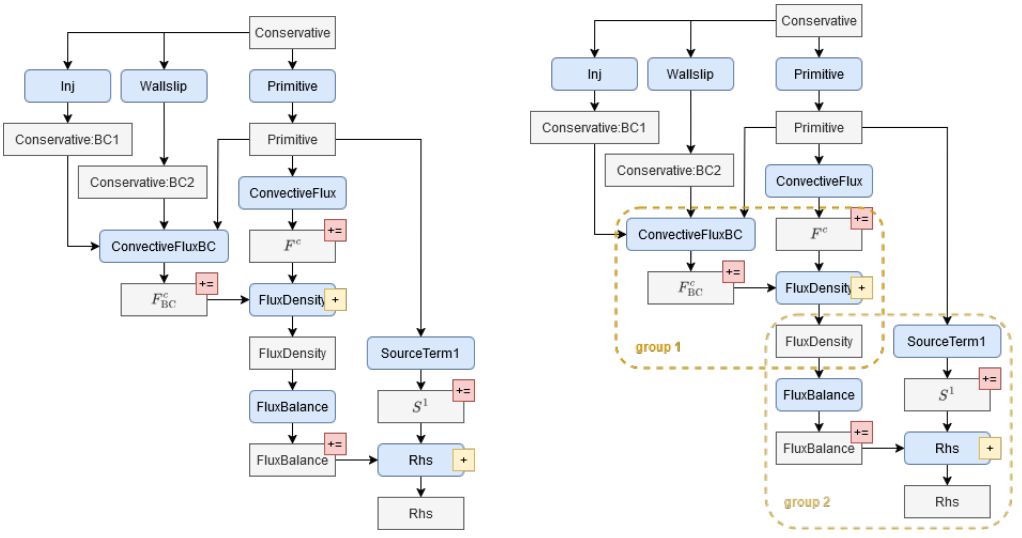
(a) Rank computation **(b)** Communicating operator graph

Figure 6. Distributed computation: analysis and transformation of the operator graph.

Using this information, the pass computes the required and provided rank for every data in the operator graph. This computation set the ranks of the inputs and outputs of the graph (i.e., `Rhs` and `Conservative`) and propagate the constraints given by the stencil information in order to minimize the number of data where the provided rank is strictly less than the required rank, i.e., data that would require a communication. Figure 6(a) shows the computed rank for our example graph (if the required and provided rank are the same, only one value is given). Here, almost all data have the same provided and required rank, except `Conservative` which is provided with a rank 0 and required with a rank 1, due to `ConvectiveFlux` being a stencil operation. Considering that the zone is distributed on two computation nodes, this rank discrepancy triggers: (i) ensuring that the other node does provide the `Conservative` data; and (ii) add to the graph the necessary communication operators to ensure that the needed rank is available before the requiring operator is executed. The resulting graph is shown in Figure 6(b): a `send` operation is added to provide the required `Conservative` halo to the other node, and a `recv` operation is added before `Primitive` so it can perform its task on a `Conservative` data with a halo of size 1.

2.5.2. *Legacy optimization*

This pass implements a general mechanism to enable a memory and computation optimization that were implemented in *elsA* and is still part of the hand-written operators in SoNICS. This optimization relies on the fact that some operators perform a simple arithmetic operation on their inputs, e.g., in Figure 3, the `Rhs` and `FluxDensity` operators simply sum their inputs into their output data. To illustrate the optimization, let us focus on the `FluxDensity` operator: here, instead of computing F^c_{BC} and F^c independently, storing them into different arrays and then summing them, maybe it is possible to use one unique array initialized to 0 and having directly the `ConvectiveFluxBC` and `ConvectiveFlux` increment this array in sequence. This is only possible if these two operators have the possibility to perform these increments. This pass thus implements an analysis to check if this optimization is possible, and then transform the graph in consequence.

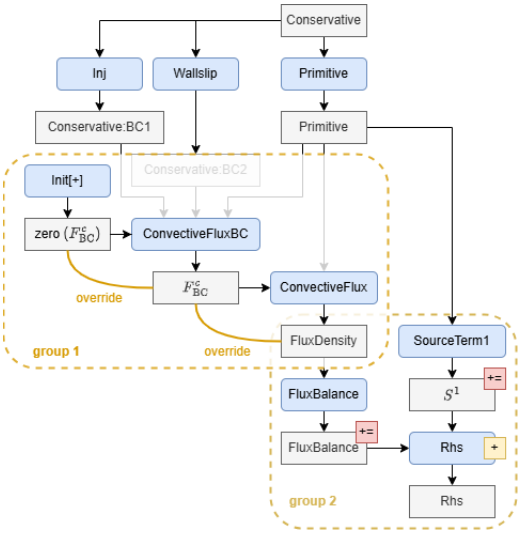


(a) Arithmetic operation information **(b)** Group analysis

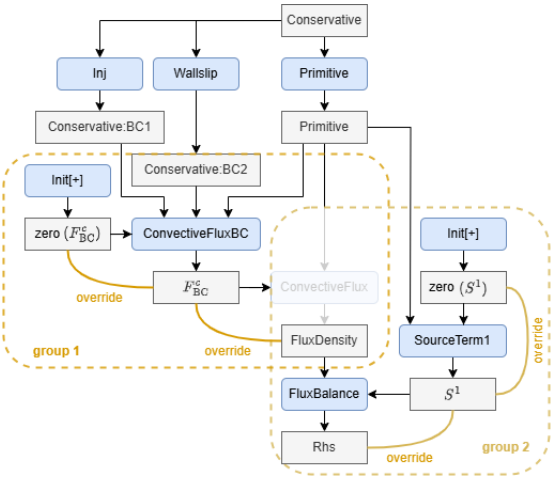
Figure 7. Legacy optimization: analysing the operator graph.

Let us illustrate this pass on the operator graph in Figure 3. The starting point of the pass is the operator graph with some information stating which operator is performing a simple arithmetic operation, and which operator can integrate this operation into an assignment. Like for the previous pass, this information is provided either by the developer or directly inferred from the operator's implementation when possible. Figure 7(a) presents the information for all the operators in the graph. Here, we can see that the operators `FluxDensity` and `Rhs` only perform a sum operation, and the operators `ConvectiveFluxBC`, `ConvectiveFlux`, `FluxBalance` and `SourceTerm1` can integrate the sum operation into an assignment. The pass then identifies the groups of operators that could be transformed into a sequence using only one array. Figure 7(b) shows that in our graph, two groups are identified: one containing the operators `FluxDensity`, `ConvectiveFluxBC` and `ConvectiveFlux`; and one containing the operators `Rhs`, `FluxBalance` and `SourceTerm1`. Finally, Figure 8 illustrates how these groups are transformed by the pass. First, Figure 8(a) presents how the group 1 is transformed. First, a new data `zero(`F^c_{BC}`)` is added, initialized to zero by the operator `Init[+]`. This array is added as input to the `FluxBC` operator which adds F^c_{BC} to it (the fact that the array is modified inplace is modeled with the `override`

edge between the data). Then, this array is given as input to `ConvectiveFlux`, which adds F_c to it: after this sequence, the data contained in this array is equal to $0+F^c_{\text{BC}}+F_c$, i.e., to `FluxDensity`. Consequently, the operator `FluxDensity` is removed from the graph, and our array is directly given as input to `FluxBalance`. In Figure 8(b), the group 2 is transformed in a similar way as the group 1, with a new array `zero`(S^1) initialize to 0, first updated by the operator `SourceTerm1`, then updated again by the operator `FluxBalance`, resulting in the sum $0+S^1+$`FluxBalance`, i.e., the `Rhs` data.



(a) Group 1 transformation



(b) Group 2 transformation

Figure 8. Legacy optimization: transforming the operator graph.

2.5.3. *Graph differentiation*

This pass manages the differentiation of the graph, and since the graph does not contain any complex control flow, it is based on the method developed for the automatic source transforma-

tion [12,13] to differentiate the graph, both in forward and backward modes. This method, applied to a graph and considering that the user wants to produce the derivative $\partial D_1/\partial D_2$, can be summed up as the three following steps.

(1) *Operator Identification*: this step identifies every operators used to compute D_1 from D_2, with their inputs and outputs involved in this computation.
(2) *Operator Differentiation*: for all of the identified operators, this step automatically generates (using automatic source transformation [12,13]) a new operator producing the derivative of the identified outputs w.r.t. the identified inputs.
(3) *Assembly*: this step combines these new operators together witht the ones already present in the operator graph to obtain a complete computation of the desired derivative.

Between the forward and backward mode, only the *Operator Differentiation* step changes.

Let us illustrate this pass on the operator graph in Figure 8(b)[1] by requesting the forward derivative ∂`Rhs`/∂`Conservative`. The *Operator Identification* step thus collects all operators between `Rhs` and `Conservative`, i.e., all operators of the graph, with all their inputs and outputs. Then, the *Operator Differentiation* step produces a derivative for every operator in the graph. We distinguish between the forward and the backward mode.

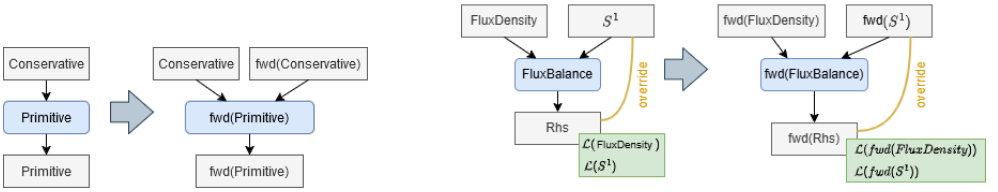
Forward mode. In forward mode, the derivative of an operator can be generated from a set of four rules. If we consider O_d an operator and O_{fwd} its forward derivative, with d_o being any derived output and d_i any deriving input, the rules can be formulated as follows:

(1) the forward derivative of d_o (resp. d_i) must be an output (resp. an input) of O_{fwd};
(2) if d_i is used in a non-linear manner in defining d_o, then d_i must be an input of O_{fwd};
(3) if d_o is stored in the same array as d_i, then the forward derivative of d_o is stored in the same array as the forward derivative of d_i.

Figure 9 illustrates these rules by presenting how the derivative of `Primitive` and `FluxBalance` are produced. First, `Primitive` is an operator taking `Conservative` in input and producing `Primitive` in output, without any additional properties. So its derivative, named `fwd(Primitive)` produces in output the derivative of `Primitive` (called `fwd(Primitive)`) and takes in parameter both the original inputs of `Primitive` (i.e., `Conservative`) and the derivative of these inputs (i.e., `fwd(Conservative)`). The `FluxBalance` operator takes `FluxDensity` and S^1 in input, and produces `Rhs` in output, with the additional property that `Rhs` is stored in the same array as S^1, and that the computation of `Rhs` is linear w.r.t. both of the operator's inputs (noted $\mathscr{L}$(`FluxDensity`), $\mathscr{L}(S^1)$). Since the usage of the operator's inputs is linear, its derivative `fwd(FluxBalance)` does not need the direct values in input, and only uses `fwd(FluxDensity)` and `fwd(`S^1`)` to produce `fwd(Rhs)`. Moreover, the memory usage pattern is the same between the direct operator and its derivative: `fwd(FluxBalance)` uses the array storing `fwd(`S^1`)` to put its `fwd(Rhs)` output.

Finally, the *Assembly* step produces a complete operator graph computing the requested derivative, which is presented in Figure 10. To improve readability, the differentiated operators have a pink color, and edges connecting derivative data are also pink. We can notice that some direct operators present in Figure 8(b) are not present anymore in this graph: this is due to the fact that some operators are linear and so do not need the direct data in input, which thus does not need to be computed.

[1] Since the derivation process is order-sensitive, it must be applied after the pass `Legacy Optimization` which reorders the operators.



(a) Derivative of `Primitive` **(b)** Derivative of `FluxBalance`

Figure 9. Graph differentiation: example of operator derivative for forward mode.

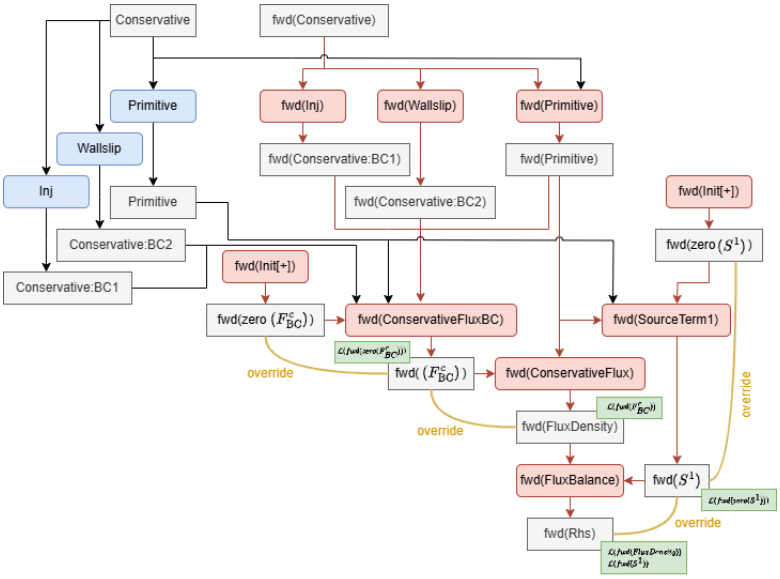


Figure 10. Graph differentiation: computing ∂`Rhs`/∂`Conservative` in forward mode.

Backward mode. In backward mode, the set of rules to produce the derivatives are more complex than in the forward mode and lead to a bigger operator graph. This complexity is due to the fact that the information flow in the backward derivative is reversed w.r.t. the direct graph. Hence, if some data is used by several operators in the direct graph, the backward derivative of this data is constructed by the sum of the contributions of all these operators' backward derivative. To deal with the fact that a direct data can have multiple contributors in backward mode, we tag these backward data with the name of the contributing operator. If we consider O_d an operator and O_{bwd} its backward derivative, with d_o being any derived output and d_i any deriving input, the rules can be formulated as follows:

(1) the backward derivative of d_o (resp. d_i) must be an input (resp. an output) of O_{bwd};
(2) if d_i is used in a non-linear manner in defining d_o, then d_i must be an input of O_{bwd};
(3) if d_o is stored in the same array as d_i, then the backward derivative of d_i is stored in the same array as the backward derivative of d_o;
(4) if d_i is not overridden by any output of O_d, then its backward derivative supports incrementation (see Section 2.5.2).

Figure 11 illustrates these rules by presenting how the derivative of `ConvectiveFluxBC`, `ConvectiveFlux` and `Primitive` are produced. First, `ConvectiveFlux` is an operator tak-

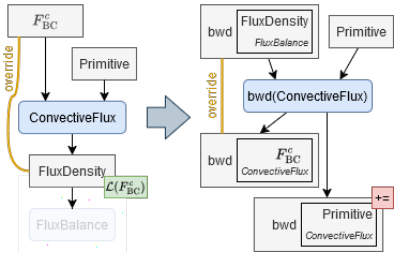
ing F^c_{BC} and `Primitive` in parameter and returning `FluxDensity` in output, with the additional properties that `FluxDensity` is stored in the same array as F^c_{BC} and is linear w.r.t. the same value. Note that the output `FluxDensity` is used only by the operator `FluxBalance` (which is thus the only contributor to the backward derivative of `FluxBalance`). Consequently, the backward derivative of `ConvectiveFlux`, called `bwd(ConvectiveFlux)`, takes in parameter the backward derivative of `FluxDensity`, called `bwd(FluxDensity)` and tagged only with `FluxBalance`, and also `Primitive` since `FluxDensity` is not linear w.r.t. this data. In output, `bwd(ConvectiveFlux)` produces the backward derivative of F^c_{BC} and `Primitive`, both tagged with `ConvectiveFlux`. Moreover, since `FluxDensity` is stored in the same array as F^c_{BC}, `bwd(`F^c_{BC}`)` is stored in the same array as `bwd(FluxDensity)`; and since `Primitive` is not overridden by any output of `ConvectiveFlux`, `bwd(Primitive)` supports incrementation. The construction of the backward derivative of `ConvectiveFluxBC` is similar, except that since it has more inputs and no linear or overriding properties, the resulting operator has more inputs and outputs, and all its outputs supports incrementation. The construction of the backward derivative of `Primitive` concludes our presentation. The primitive operator has a simple structure, with only one input and one output, with no linear or overriding properties, but its output `Primitive` is used by three different operators that will all become contributors to the backward derivative of the `Primitive` data. Hence the operator `bwd(Primtive)` has only one output, `bwd(Conservative)` which supports incrementation, and two inputs: `Conservative` (since `Primitive` has no linear properties), and the main one that collects in a sum all the contributions made to `bwd(Primitive)`.

Finally, the *Assembly* step produces a complete operator graph computing the requested derivative, which is presented in Figure 12 where direct operators have been removed for readability.

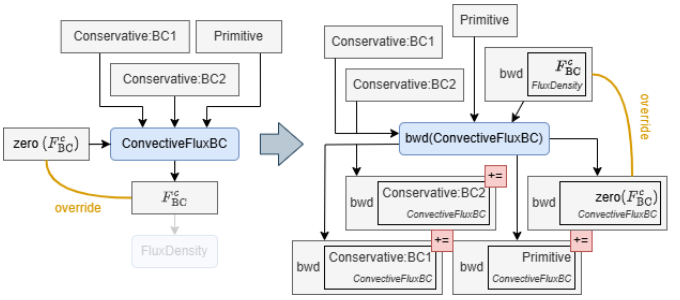
2.5.4. *Memory management*

This pass completes the graph with *memory allocation* and *array copy* operators. Indeed, all the data computed in the graph must be stored on some array which must be allocated. Some of these array must be allocated only once (for data whose size do not change over time), some other must be reallocated regularly, e.g., for iso-surfaces. Moreover, as illustrated in the *Legacy Optimization* pass, some operator may override the content of an array with its own output, possibly creating issues if that content is used by another operator. To avoid such data-race, in most case it is enough to force the execution of the overriding operator after the other ones using the data. But in rare cases, the input data must be copied. This pass thus analyses the data manipulated by the graph, its size, and computes an allocation scheme suited for it, i.e., it associates a memory space in an array for every data while ensuring that data is overwritten only when it is not used anymore. Then it adds to the graph the corresponding memory allocation operators.

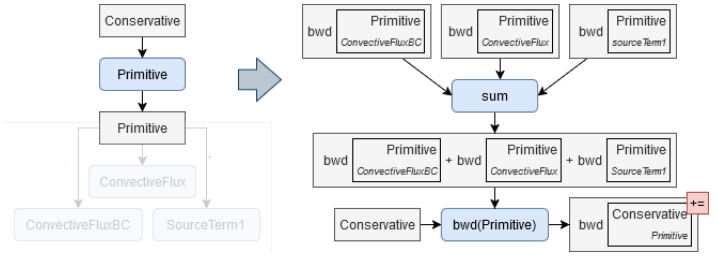
Figure 13 presents the result of the pass on the graph in Figure 8(b). Since every overwritten data is used by only one operator, no ordering between operators nor copies were added to the graph. However, five allocation operators were added to the graph: (i) the `Primitive` data is created by the `Primitive` operator (which needs an array to store it) and so that array is allocated by the operator `Alloc[Primitive]` before the `Primitive` operator is executed; (ii) and (iii) similarly for the `Inj` and `Wallslip` operators, the array where they store their output is allocated before they execute; (iv) since the data `zero(`F^c_{BC}`)`, F^c_{BC} and `FluxDensity` all share the same array, only one allocation operator is added to create the array that will be filled with 0 by the `Init[+]` operator on the left of the graph; and (v) similarly with the data `zero(`S^1`)`, S^1 and `Rhs`, only one allocation operator is added to create the array that will be filled with 0 by the



(a) Derivative of `ConvectiveFlux`



(b) Derivative of `ConvectiveFluxBC`



(c) Derivative of `Primitive`

Figure 11. Graph differentiation: example of operator derivative for backward mode.

`Init[+]` operator on the right of the graph. Note that since the `Conservative` data is an input of the graph, it does not need to be allocated.

3. Results

SoNICS and its architecture have been gradually tested and validated from simple cases like NACA012 airfoil on inviscid flow [42,43] to more complex academical case like the ONERA M6 Wing [44] and specific open usecases to validate turbulence models [45], up to industrial cases. Currently, SoNICS supports: Conducted Reynolds-Averaged Navier–Stokes (RANS) simulations with Spalart–Allmaras [46] and k-ω [47] turbulence models; Incorporated transition modeling through transport equations, specifically employing the Menter–Langtry transition model [48]; and Multi-species simulations to account for combustion phenomena. Moreover, unsteady computation with Runge–Kutta [49] or BDF2 [50] approaches and advanced version of the Zonal Detached Eddy Simulation (ZDES) based on Spalart–Allmaras model proposed by Deck [51] have also been implemented, tested and validated. Furthermore, to effectively manage computations

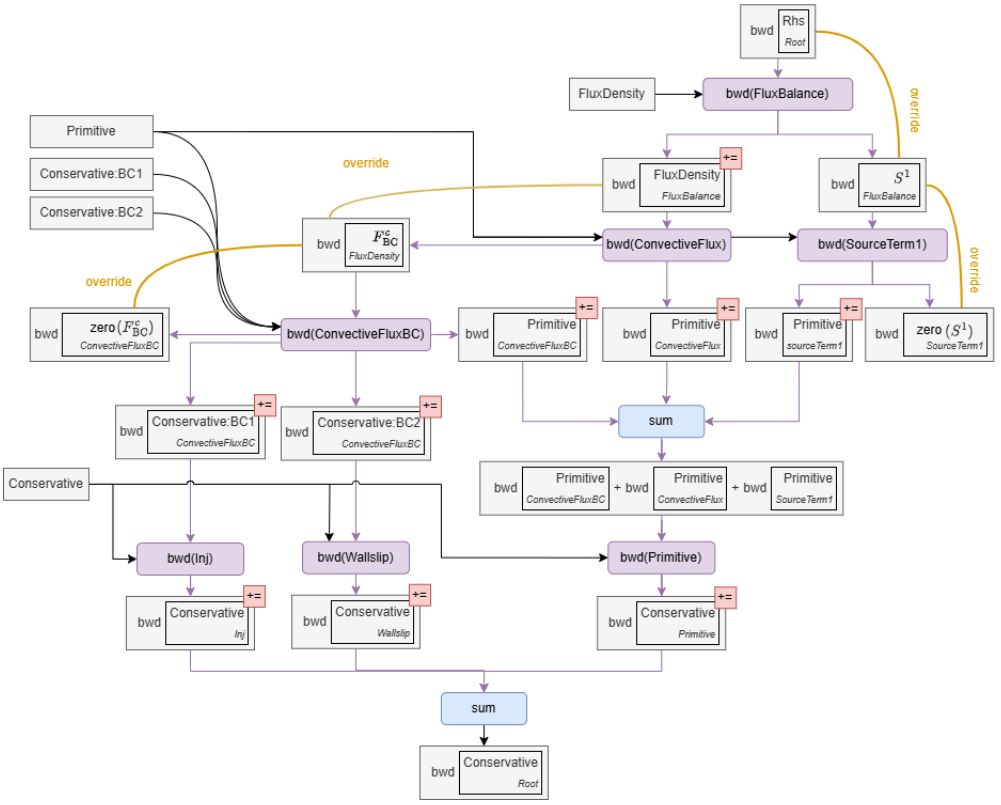


Figure 12. Graph differentiation: computing ∂`Rhs`/∂`Conservative` in backward mode.

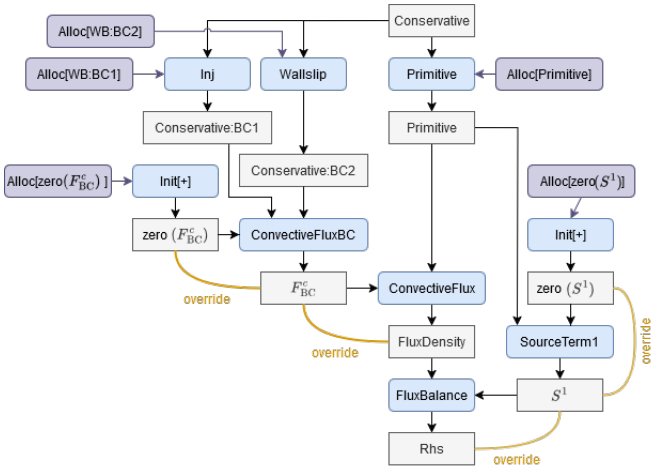


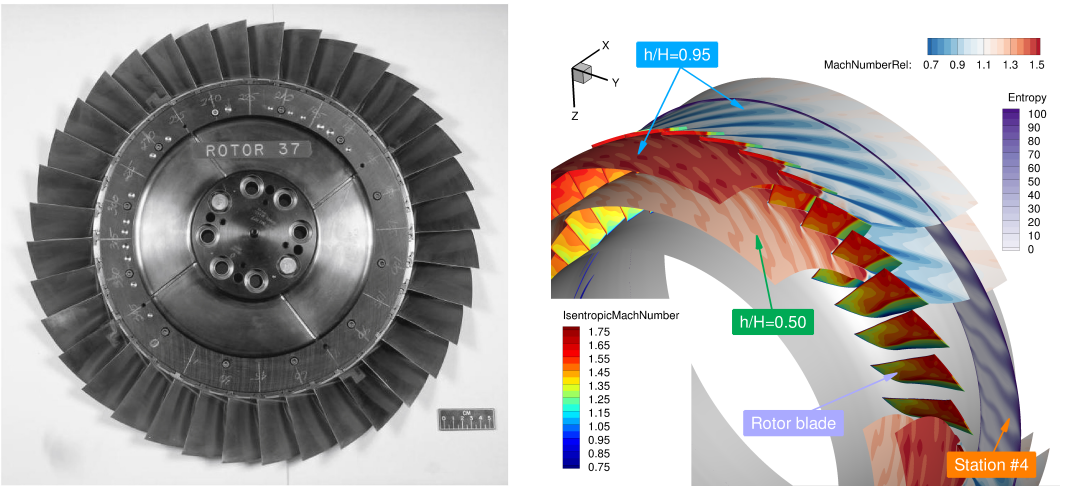
Figure 13. Memory management: adding the allocation operators.

on meshes with high anisotropy ratios, SoNICS includes a vertex-centered solver. This approach is widely recognized for its robustness when dealing with highly anisotropic meshes, ensuring accurate and stable numerical solutions even in challenging geometric configurations.

The rest of the section presents two cases of interest to highlight the different architecture choices of SoNICS: (i) the NASA Rotor 37 [52] demonstrates the capabilities of turbomachinery simulations and compares the results of SoNICS to those of *elsA*; (ii) a generic rocket after body [53,54] compares the cell-centered solver and the vertex-centered approach with mesh adaptation.

3.1. *NASA Rotor 37*

The NASA Rotor 37, presented in Figure 14, is a widely recognized and extensively studied test case in the CFD field for turbomachinery applications. This transonic axial compressor rotor was designed in the 1970s and was tested at NASA's Lewis Research Center (now Glenn Research Center) [55]. As an isolated rotor, the NASA Rotor 37 continues to be relevant in modern CFD research, serving as a platform for testing new numerical methods, turbulence models, and optimization techniques. Its well-documented geometry and experimental data make it an ideal candidate for validating CFD solvers and exploring advanced concepts in turbomachinery flow physics.



(a) NASA Rotor 37 blade

(b) Computation with SoNICS

Figure 14. NASA Rotor 37.

To validate SoNICS, a comparative study is conducted using the NASA Rotor 37. The same calculation is performed using both *elsA* and SoNICS in a nearly identical numerical configuration using a Roe scheme on an unstructured hexahedral mesh consisting of 1,480,704 cells. The turbulence is modeled using the Spalart–Allmaras (SA) model, a widely adopted one-equation Reynolds-Averaged Navier–Stokes (RANS) turbulence model. Our comparison focuses on several key parameters of interest, including the isentropic Mach number. Moreover, to showcase the flexibility of SoNICS's architecture, the comparison is carried out on two different scenarios: (i) first-order accuracy; and (ii) second-order accuracy without slope limiters.

Figure 15 shows respectively a slice of the isentropic Mach number at mid span blade (h/H = 50%), the distribution of the stagnation pressure and temperature ratio among the radius at station 4 (cf. Figure 14(b)). We can see here that the results between *elsA* and SoNICS are similar: the tiny differences are caused by the fact that turbulence model are not strictly the same (the model in SoNICS slightly improves over the one in *elsA* by taking into account compressibility phenomena).

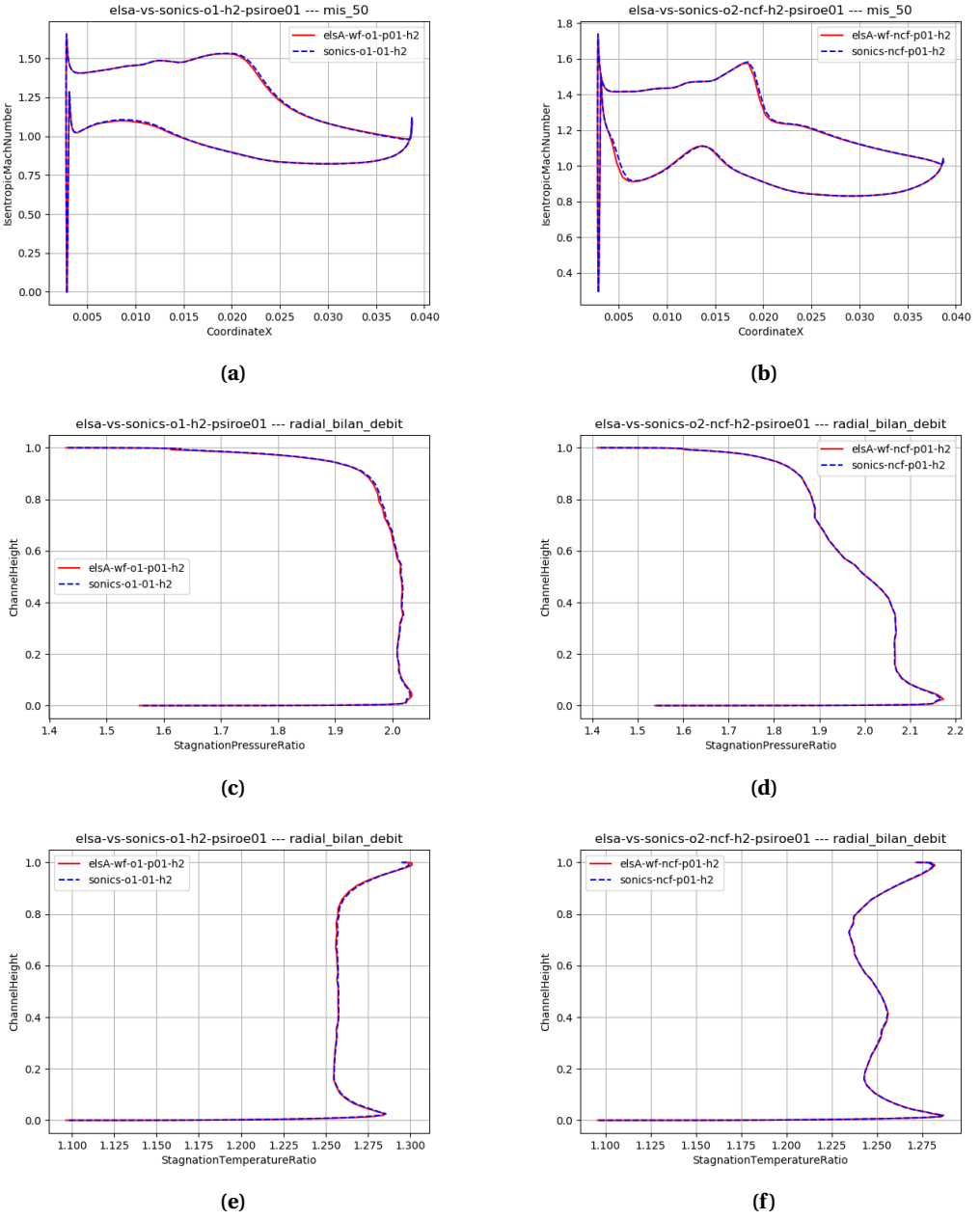


Figure 15. elsA and SoNICS comparison of isentropic Mach number (a) at order 1 and (b) order 2; stagnation pressure ratio (c) at order 1 and (d) order 2; stagnation temperature ratio (e) at order 1 and (f) order 2.

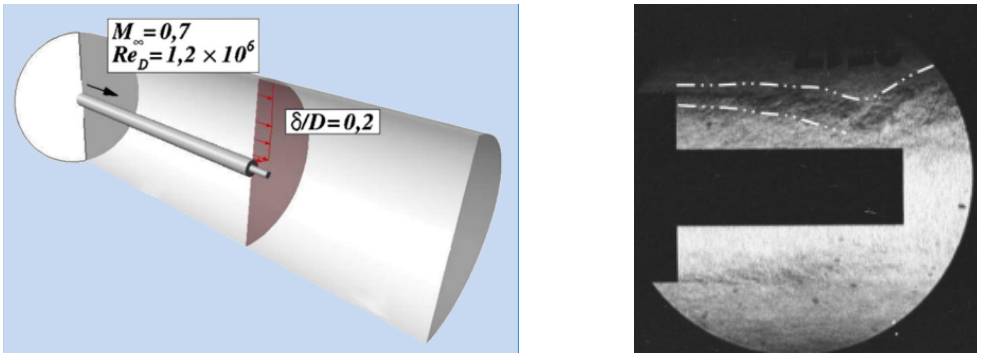
3.2. *Simplified launcher afterbody*

This case consists of two concentric cylinders with the inner cylinder modeling the nozzle as shown in Figure 16. The primary cylinder has a diameter of $D = 0.1\,\mathrm{m}$ and a length of $L = 1.2D$.

Experimental investigations were conducted in ONERA's S3Ch transonic wind tunnel [53] and advanced turbulence modelisation like Zonal Detached Eddy Simulation (ZDES) approach have been studied at ONERA on this case [54]. The experiment was conducted at a Mach number of $M = 0.7$ with a Reynolds number based on the cylinder diameter of $\mathrm{Re}_D = 1.2 \times 10^6$ and such that the boundary layer thickness at the axisymmetric step is $\delta = 0.2D$. The Reynolds number is sufficiently high to justify the use of Reynolds-Averaged Navier–Stokes (RANS) modeling for turbulence. In this particular case, the Spalart–Allmaras model is employed as the turbulence closure scheme.

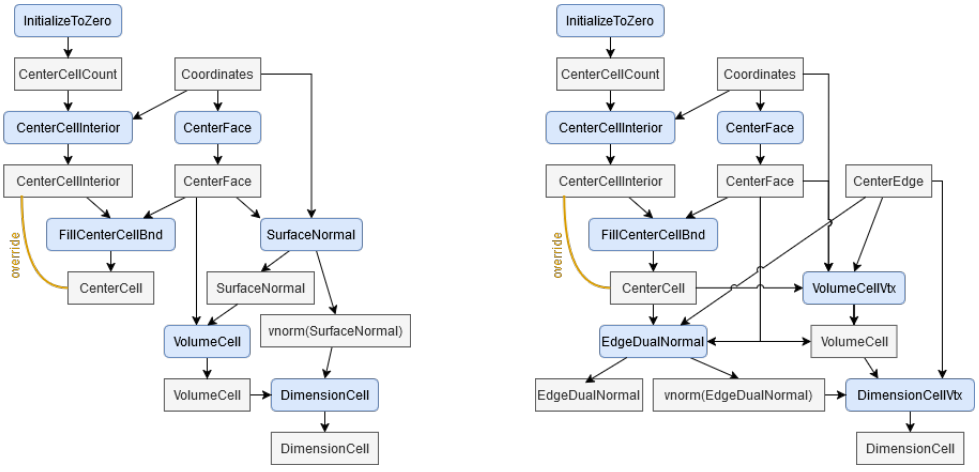
Two computations have been done on this geometry:

(1) a simulation performed with the cell-center solver on a fine hand-made hexa mesh;
(2) a simulation performed with the vertex-center solver on tetra meshes; the simulation is initiated on an extremely coarse tetrahedral mesh that was refined using a local Mach number criterion; this adaptive mesh refinement procedure was iteratively repeated until a predefined convergence criterion was satisfied.



(a) Overview extracted from [54] **(b)** Strioscopy extracted from [56]

Figure 16. Generic rocket aft body S3Ch.



(a) Cell-center solver **(b)** Vertex-center solver

Figure 17. The geometry part of the operator graph.

Figure 17 shows the flexibility of SoNICS's architecture by presenting the part of the operator graph dedicated to the computation of the geometry for both computations: while these graphs are very different, they were automatically produced and executed by SoNICS's architecture alone, with one option changed it input. Figure 18 shows the remarkable superposition between the isolines obtained from the cell-centered solver simulation and the contours resulting from the vertex-centered solver simulation after 15 adaptive refinement iterations. And finally, Figure 19 presents a comparative analysis of the pressure coefficient distributions obtained from two sources: (i) the cell-centered solver simulation; and (ii) the vertex-centered solver simulation after 15 adaptive refinement iterations. This validates the precision and reliability of the vertex-center solver w.r.t. the cell-center solver inherited from *elsA*, and moreover, not shown in this figure is the high degree of consistency between both numerical methods in this test case.

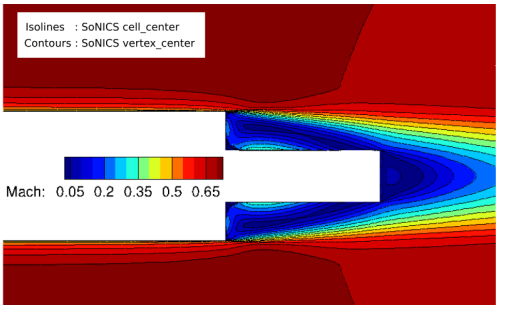


Figure 18. Isolines from the cell-centered solver simulation, superposed with the contours from the vertex-centered solver simulation after 15 adaptive refinement iterations: they are identical.

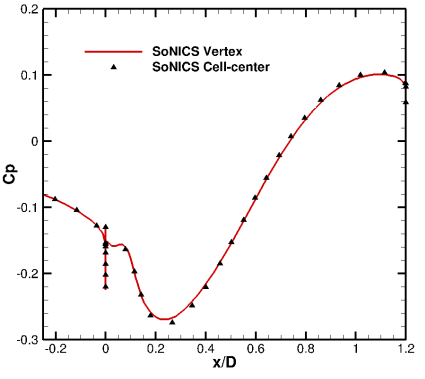


Figure 19. Pressure coefficient distributions from the cell-centered solver, and the vertex-centered solver after 15 adaptive refinement iterations.

4. Related work

Many CFD solvers have been developed over the years. However, while plenty of bibliography exists on the models supported by these tools, their simulations and performances, very little

can be found on their workflow, even for open source solvers. In the following, we present five of the most well-known solvers.

AVBP. This software [57,58] is developed at CERFACS and is specifically designed for simulating unsteady turbulent compressible and reactive flows and especially Large Eddy Simulation (LES). It is capable of massive parallel processing, supporting both CPU and GPU computations, and is implemented in Fortran.

No documentation on its workflow is directly available, but the available presentations of the tool indicate that it does not separate the control flow from its computation, which can add a large runtime overhead to the computation, as discussed in this article. Its user interface consists of a graphical interface named C3S to facilitate workflow management, and a component called HIP to handle preprocessing. Finally, since AVBP is dedicated to unsteady computation, gradient optimization does not fall within its primary area of interest, and so it does not support automatic code differentiation.

YALES2. This software [59] is developed at CORIA for simulating unsteady, turbulent, incompressible, compressible and reactive flows featuring feature-based mesh adaptation. It is composed by several solvers, each dedicated to a specific family of simulation. All of its solvers are implemented in Fortran 2008, have distributed hardware support, and some have been ported to NVIDIA and AMD GPUs.

No documentation on its workflow is directly available, but it is clear from its design that the part of the control flow dedicated to choosing the simulation model is separated from the computation, since every supported model is implemented by a dedicated solver. Moreover, the control flow dedicated to the management of the hardware and mesh topologies is most probably implemented in a similar fashion as *elsA*, using a factory identifying what to compute before actually starting the simulation. Its user interface consists of a python API that utilizes the f90wrap library to facilitate the various Fortran data types wrapping. Finally, YALES2 is dedicated to unsteady computation and does not support automatic code differentiation.

CODA. This software [60] (previously called *Flucs*) is a collaboration between ONERA, DLR and Airbus and is implemented in C++, using templates for performance optimization. Like *elsA* and SoNICS, it supports a wide range of physical models including (but not limited to): steady-state Reynolds-Averaged Navier–Stokes equations (RANS) with various turbulence models, Large Eddy Simulation (LES), Hybrid RANS/LES models for complex flow scenarios, Multi-species and reactive flows. One noticeable particularity is that CODA implements a second-order finite-volume method and higher-order Discontinuous Galerkin (DG) methods tailored on unstructured grids. Moreover, it supports distributed hardware (based on the MPI [61], GASPI [62] and OpenMP [63] libraries), and while it does not completely support GPUs, its implicit phase is based on the Spliss library [64] which has full GPU support.

The control flow of CODA is split in two parts: the model and schema are managed during compilation, producing one solver for a dedicated family of simulations; and before performing the simulation, a factory identifies what to compute from the hardware and mesh topologies. As user interface, it provides a python API. Automatic Differentiation is implemented in CODA with ADOL-C [65] which performs a dynamic analysis during the simulation, with a possible memory overhead as discussed in [20].

OpenFOAM. This software [66] is a free, open-source solver suite with a modular architecture similar to YALES2 and CODA. It supports a wide range of physical models implemented by around 100 different solvers, and can be executed on distributed hardware, including GPUs [67]. One noticeable particularity is that OpenFOAM's architecture allows for easy coupling between its different solvers.

The control flow of OpenFOAM is split in two parts: the model and schema are managed during compilation, producing one solver for a dedicated family of simulations; and before performing the simulation, a factory identifies what to compute from the hardware and mesh topologies. Its user interface is structured as multiple configuration files that the user need to fill using a dedicated syntax. Finally, a specialized and partial version of OpenFOAM, called DAFoam [68], uses dynamic analysis to achieve Automatic Differentiation based on CoDiPack [16] and MeDiPack [69].

SU2. This software [70] is a free, open-source solver suite implemented in C++ dedicated to diverse fluid dynamics problems, including steady and unsteady simulations, compressible and incompressible flows, turbulence modeling and multi-physics problems. All of its solvers can be run on distributed hardware, but support for GPUs is still partial. SU2's code base was designed for easy customization and integration of new features.

The control flow of SU2 is split in two parts, like the one of OpenFOAM. A python-based graphical interface, called SU2GUI [71] is provided to the user to configure and manage SU2. Finally, Automatic Differentiation is implemented in SU2 with dynamic analysis using CoDiPack.

Although many articles around these different solvers emphasize numerical methods or different physical modeling, information that directly addresses the architectural details of these solvers is difficult to find. However, as discussed for each solver individually, it seems that like *elsA*, they all offer an architecture where user choices are directly transformed into a static list of functions to be executed at runtime. The particularity of SoNICS's architecture is its capability to first generate a graph of operators from the user choices prior to the code execution. This graph can be analyzed and transformed to optimize the runtime execution, similarly to a compiler analysing and optimizing the code in input to produce an efficient executable. Moreover, thanks to SoNICS's code generation capabilities, the implementation of every variant of each operator can be also generated on the fly.

It is important to emphasize that SoNICS's operator graph, its manipulation and distribution are largely inspired by Artificial Intelligence (AI) frameworks like TensorFlow [72] and PyTorch [73]. This alignment is a deliberate choice driven by shared constraints and objectives, like efficiency and portablity. Moreover, modern AI frameworks include Automatic Differentiation capabilities, typically implemented through sophisticated source code transformation tools like Enzyme [13].

5. Conclusion

In this article, we presented the design choices and the structure of the new SoNICS CFD solver. All of these choices were carefully made to satisfy the different expectations a user could have. We illustrated our design with several examples to show how a user configuration is translated, step by step, into an operator graph containing all the information necessary to perform a computation answering the user's request, with a focus on memory and time efficiency. Moreover, we applied the complete tool on two important usecases, the *Rotor 37* and the *Simplified launcher afterbody*, validating its capability to seamlessly manage complex configurations and different hardwares to perform both a direct simulation and a mesh refinement task.

In future work, we intend to improve and add several elements of the workflow. For instance, the `Graph Differentiation` pass (in Section 2.5.3) is still in development; the `Memory Management` pass (in Section 2.5.4) could be improved so more arrays could be reused, thus improving the memory footprint of SoNICS; the `Builder` task (in Section 2.2) can be refactored; and the `Distribution` task (in Section 2.2) could be changed to compute a better load-balancing, using for instance algorithms of the HEFT family [74–76]. Moreover, more memory and time optimizations can be added to the SoNICS's workflow: for instance, operators looping on the same

data could be merged if they execute on CPU, so they would share the same loop, thus reducing the latency due to cache misses. Finally, we also intend to perform more simulations using SoNICS, including coupling it with other solvers, to check if its current API is satisfactory or should be improved to satisfy every user's needs.

Acknowledgments

The authors would like to thank Julien Husson, Julien Marty and Bruno Maugars for their work on the usecases presented in this article.

Declaration of interests

The authors do not work for, advise, own shares in, or receive funds from any organization that could benefit from this article, and have declared no affiliations other than their research organizations.

References

[1] L. Cambier, M. Gazaix, S. Heib, S. Plot, M. Poinot, J. P. Veuillot, J.-F. Boussuge and M. Montagnac, "An overview of the multi-purpose *elsA* flow solver", *Aerospace Lab* (2011), no. 2, article no. AL02-10 (15 pages).

[2] L. Cambier, S. Heib and S. Plot, "The Onera elsA CFD software: input from research and feedback from industry", *Mech. Ind.* **14** (2013), no. 3, pp. 159–174.

[3] S. Plot, "The High Level of Maturity of the elsA CFD Software for Aerodynamics Applications", in *EUCASS 2019*, 2019.

[4] W. Thollet, G. Dufour, X. Carbonneau and F. Blanc, "Body-force modeling for aerodynamic analysis of air intake–fan interactions", *Int. J. Numer. Methods Heat Fluid Flow* **26** (2016), no. 7, pp. 2048–2065.

[5] G. Carrier, D. Destarac, A. Dumont, et al., "Gradient-based aerodynamic optimization with the elsA software", in *52nd Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2014.

[6] B. G. van Der Wall, C. Kessler, Y. Delrieux, P. Beaumier, M. Gervais, J. C. Hirsch, K. Pengel and P. Crozier, "From aeroacoustics basic research to a modern low-noise rotor blade", *J. Am. Helicopter Soc.* **62** (2017), no. 4, pp. 1–16.

[7] E. R. Leon, A. Le Pape, M. Costes, J. A. Désidéri and D. Alfano, "Concurrent aerodynamic optimization of rotor blades using a Nash game method", *J. Am. Helicopter Soc.* **61** (2016), no. 2, pp. 1–13.

[8] D. Guegan, M. Schvallinger, F. Julienne, N. Gourdain and M. Gazaix, "Three-dimensional full annulus unsteady RANS simulation of an integrated propulsion system", in *51st AIAA/SAE/ASEE Joint Propulsion Conference*, American Institute of Aeronautics and Astronautics, 2015.

[9] T. Berthelon, A. Dugeai, J. Langridge and F. Thouverez, "Ground effect on fan forced response", in *Proc. of the 15th International Symposium on Unsteady Aerodynamics, Aeroacoustics and Aeroelasticity of Turbomachines, Vol. ISUAAAT15-094, American Society of Mechanical Engineers*, 2018.

[10] F. Blondel, M. Stanciu, F. Lebœuf and M. Lance, "Modelling unsteadinesses and polydispersion in wet steam flows using the quadrature method of moments and a two-equation model", in *Proceedings of the 10th European Conference on Turbomachinery Fluid dynamics & Thermodynamics*, 2013.

[11] M. Stanciu, M. Marcelet and J.-M. Dorey, "Numerical Investigation of Condenser Pressure Effect on Last Stage Operation of Low Pressure Wet Steam Turbines", in *ASME Turbo Expo 2013: Turbine Technical Conference and Exposition. Volume 5B: Oil and Gas Applications; Steam Turbines*, ASME Press, 2013, pp. 1–11.

[12] L. Hascoët and V. Pascual, "The Tapenade automatic differentiation tool: Principles, model, and specification", *ACM Trans. Math. Softw.* **39** (2013), no. 3, article no. 20 (43 pages).

[13] W. S. Moses, S. H. K. Narayanan, L. Paehler, V. Churavy, M. Schanen, J. Hückelheim, J. Doerfert and P. Hovland, "Scalable automatic differentiation of multiple parallel paradigms through compiler augmentation", in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, IEEE Press: Dallas, Texas, 2022, (18 pages).

[14] D. Maclaurin, D. Duvenaud and R. P. Adams, "Autograd: effortless gradients in numpy", in *ICML 2015 AutoML Workshop*, 2015.

[15] J. Bradbury, R. Frostig, P. Hawkins, et al., *JAX: composable transformations of Python+NumPy programs*, version 0.3.13, 2018. Online at http://github.com/jax-ml/jax (accessed on October 31, 2025).

[16] M. Sagebaum, T. Albring and N. R. Gauger, "High-Performance Derivative Computations using CoDiPack", *ACM Trans. Math. Softw.* **45** (2019), no. 4, article no. 38 (26 pages).

[17] A. Haj-Ali, N. K. Ahmed, T. Willke, Y. S. Shao, K. Asanovic and I. Stoica, "NeuroVectorizer: end-to-end vectorization with deep reinforcement learning", in *CGO '20: Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization* (J. Mars, L. Tang, J. Xue and P. Wu, eds.), ACM Press: San Diego, CA, USA, 2020, pp. 242–255.

[18] Y. Chen, C. Mendis and S. Amarasinghe, "All you need is superword-level parallelism: systematic control-flow vectorization with SLP", in *PLDI 2022: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (R. Jhala and I. Dillig, eds.), ACM Press: San Diego, CA, USA, 2022, pp. 301–315.

[19] A. H. Tabar, R. Bubel and R. Hähnle, "Automatic loop invariant generation for data dependence analysis", in *FormaliSE '22: Proceedings of the IEEE/ACM 10th International Conference on Formal Methods in Software Engineering* (S. Gnesi, N. Plat, A. Hartmanns and I. Schaefer, eds.), ACM Press, 2022, pp. 34–45.

[20] B. Maugars, S. Bourasseau, C. Content, B. Michel, B. Berthoul, J. N. Ramirez, P. Raud and L. Hascoët, "Algorithmic Differentiation for an efficient CFD solver", in *ECCOMAS 2022: 8th European Congress on Computational Methods in Applied Sciences and Engineering*, 2022.

[21] P. Clements and L. Northrop, *Software product lines: practices and patterns*, Addison Wesley Longman, 2001.

[22] D. Poirier, S. R. Allmaras, D. McCarthy, M. Smith and F. Enomoto, "The CGNS system", in *29th AIAA, Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, 1998.

[23] M. Poinot and C. L. Rumsey, "Seven keys for practical understanding and use of CGNS", in *2018 AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2018.

[24] J. Coulet, C. Benazet, B. Berthoul and B. Maugars, *Maia: a Python and C++ library for parallel algorithms and manipulations over CGNS meshes*, version 1.5, 2024. Online at https://github.com/onera/maia/ (accessed on October 31, 2025).

[25] G. Kiczales, "Aspect-oriented programming", *ACM Comput. Surv.* **28** (1996), no. 4es, article no. 154.

[26] G. Kiczales, "AspectJ(tm): Aspect-Oriented Programming in Java", in *Objects, Components, Architectures, Services, and Applications for a Networked World, International Conference NetObjectDays, NODe 2002, Erfurt, Germany, October 7-10, 2002, Revised Papers* (M. Aksit, M. Mezini and R. Unland, eds.), Lecture Notes in Computer Science, Springer, 2002, p. 1.

[27] G. Kiczales, "Aspect-oriented programming", in *ICSE '05: Proceedings of the 27th International Conference on Software Engineering* (G.-C. Roman, W. G. Griswold and B. Nuseibeh, eds.), ACM Press, 2005, p. 730.

[28] F. Pellegrini and J. Roman, "SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs", in *High-Performance Computing and Networking, International Conference and Exhibition, HPCN Europe 1996, Brussels, Belgium, April 15-19, 1996, Proceedings* (H. M. Liddell, A. Colbrook, L. O. Hertzberger and P. M. A. Sloot, eds.), Lecture Notes in Computer Science, Springer, 1996, pp. 493–498.

[29] R. Barat, C. Chevalier and F. Pellegrini, "Multi-criteria Graph Partitioning with Scotch", in *Proceedings of the Eighth SIAM Workshop on Combinatorial Scientific Computing, CSC 2018, Bergen, Norway, June 6-8, 2018* (F. Manne, P. Sanders and S. Toledo, eds.), Society for Industrial and Applied Mathematics, 2018, pp. 66–75.

[30] G. Karypis, K. Schloegel and V. Kumar, "ParMETIS — Parallel graph partitioning and sparse matrix ordering library", version 3.1, 2003. Online at http://charm.cs.uiuc.edu/users/gupta/2012_CloudHPCLB_charm/src/libs/ck-libs/parmetis/Manual/manual.pdf.

[31] G. Karypis, "METIS and ParMETIS", in *Encyclopedia of Parallel Computing* (D. A. Padua, ed.), Springer, 2011, pp. 1117–1124.

[32] E. Quemerais, B. Maugars and B. Andrieu, *ParaDiGM: a library for parallel computational geometry*, version 2.6.0, 2024. Online at https://github.com/onera/paradigm/ (accessed on October 31, 2025).

[33] T.-W. Huang, D.-L. Lin, C.-X. Lin and Y. Lin, "Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System", *IEEE Trans. Parallel Distrib. Syst.* **33** (2022), no. 6, pp. 1303–1320.

[34] D. Romero-Organvidez, P. N. Ayuso, J. A. Galindo and D. Benavides, "Kconfig metamodel: a first approach", in *Proceedings of the 28th ACM International Systems and Software Product Line Conference - Volume B, SPLC 2024, Dommeldange, Luxembourg, September 2-6, 2024* (M. Cordy, D. Strüber, M. Pinto, et al., eds.), ACM Press, 2024, pp. 55–60.

[35] R. Schröter, S. Krieter, T. Thüm, F. Benduhn and G. Saake, "Feature-model interfaces: the highway to compositional analyses of highly-configurable systems", in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016* (L. K. Dillon, W. Visser and L. A. Williams, eds.), ACM Press, 2016, pp. 667–678.

[36] I. Schaefer, L. Bettini, V. Bono, F. Damiani and N. Tanzarella, "Delta-Oriented Programming of Software Product Lines", in *Software Product Lines: Going Beyond - 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Proceedings* (J. Bosch and J. Lee, eds.), Lecture Notes in Computer Science, Springer, 2010, pp. 77–91.

[37] M. Lienhardt, "PYDOP: A Generic Python Library for Delta-Oriented Programming", in *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B, SPLC 2023, Tokyo, Japan, 28 August 2023- 1 September 2023* (P. Arcaini, M. H. t. Beek, G. Perrouin, et al., eds.), ACM Press, 2023, pp. 30–33.

[38] A. Meurer, C. P. Smith, M. Paprocki, et al., "SymPy: symbolic computing in Python", *PeerJ Comput. Sci.* **3** (2017), article no. e103 (27 pages).

[39] M. Bauer, J. Hötzer, D. Ernst, et al., "Code generation for massively parallel phase-field simulations", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019* (M. Taufer, P. Balaji and A. J. Peña, eds.), ACM Press, 2019, (32 pages).

[40] M. Lienhardt, M. H. t. Beek and F. Damiani, "Product lines of dataflows", *J. Syst. Software* **210** (2024), article no. 111928 (22 pages).

[41] M. Lienhardt, "The Hrewrite Library: A Term Rewriting Engine for Automatic Code Assembly", in *Rewriting Logic and Its Applications - 15th International Workshop, WRLA 2024, Luxembourg City, Luxembourg, April 6-7, 2024, Revised Selected Papers* (K. Ogata and N. Martí-Oliet, eds.), Lecture Notes in Computer Science, Springer, 2024, pp. 165–178.

[42] K. W. McAlister, L. W. Carr and W. J. McCroskey, *Dynamic stall experiments on the NACA 0012 airfoil*, Technical Publication, NASA, no. 19780009057, 1978.

[43] W. J. McCroskey, *A critical assessment of wind tunnel results for the NACA 0012 airfoil*, Technical Memorandum, NASA, no. 19880002254, 1987.

[44] V. Schmitt and F. Charpin, *Pressure distributions on the ONERA-M6-Wing at transonic Mach numbers*, techreport, ONERA, no. AGARD AR 138, 1979.

[45] Turbulence Model Benchmarking Working Group, *Turbulence Modeling Resource*, June 3, 2025. Online at https://turbmodels.larc.nasa.gov/ (accessed on October 30, 2025).

[46] P. R. Spalart and S. R. Allmaras, "A one-equation turbulence model for aerodynamic flows", *Rech. Aerosp.* **42** (1994), no. 1, pp. 5–21.

[47] F. R. Menter, "Two-Equation Eddy-Viscosity Turbulence Models for Engineering Applications", *AIAA J.* **32** (1994), no. 8, pp. 1598–1605.

[48] R. B. Langtry and F. R. Menter, "Correlation-Based Transition Modeling for Unstructured Parallelized Computational Fluid Dynamics Codes.", *AIAA J.* **47** (2009), no. 12, pp. 2894–2906.

[49] J. H. Williamson, "Low-Storage Runge-Kutta Schemes.", *J. Comput. Phys.* **35** (1980), pp. 48–56.

[50] C. W. Gear, *Numerical initial value problems in ordinary differential equations*, Prentice Hall, 1971.

[51] S. Deck, "Recent improvements in the Zonal Detached Eddy Simulation (ZDES) formulation", *Theor. Comput. Fluid Dyn.* **26** (2012), pp. 523–550.

[52] S. Mouriaux, F. Bassi, A. Colombo and A. Ghidoni, "NASA Rotor 37", in *TILDA: Towards Industrial LES/DNS in Aeronautics: Paving the Way for Future Accurate CFD - Results of the H2020 Research Project TILDA, Funded by the European Union, 2015 -2018* (C. Hirsch, K. Hillewaert, R. Hartmann, V. Couaillier, J.-F. Boussuge, F. Chalot, S. Bosniakov and W. Haase, eds.), Springer, 2021, pp. 533–544.

[53] D. Deprés, P. Reijasse and J. P. Dussauge, "Analysis of unsteadiness in afterbody transonic flows", *AIAA J.* **42** (2004), no. 12, pp. 2541–2550.

[54] R. Pain, P.-E. Weiss and S. Deck, "Zonal Detached Eddy Simulation of the Flow Around a Simplified Launcher Afterbody", *AIAA J.* **52** (2014), no. 9, pp. 1967–1979.

[55] L. Reid and R. D. Moore, *Design and overall performance of four highly loaded, high-speed inlet stages for an advanced high-pressure-ratio core compressor*, Technical Paper, NASA, no. NASA-TP-1337, 1978.

[56] P. Reijasse, *Étude expérimentale de l'écoulement d'arrière-corps du lanceur Ariane 5 dans la soufflerie transsonique S3Ch*, techreport, ONERA, 2007.

[57] CERFACS, *AVBP*, version 7.15, 2025. Online at https://avbp-wiki.cerfacs.fr/ (accessed on October 31, 2025).

[58] T. Schonfeld and M. Rudgyard, "Steady and unsteady flow simulations using the hybrid flow solver AVBP", *AIAA J.* **37** (1999), no. 11, pp. 1378–1385.

[59] V. Moureau, P. Domingo and L. Vervisch, "Design of a massively parallel CFD code for complex geometries", *Comptes Rendus. Mécanique* **339** (2011), no. 2–3, pp. 141–148.

[60] T. Leicht, J. Jägersküpper, D. Vollmer, A. Schwöppe, R. Hartmann, J. Fiedler and T. Schlauch, "DLR-Project Digital-X — Next Generation CFD Solver 'Flucs'", in *Deutscher Luft- und Raumfahrtkongress 2016*, 2016, (14 pages).

[61] F. Nielsen, "Introduction to MPI: The Message Passing Interface", in *Introduction to HPC with MPI for Data Science*, Undergraduate Topics in Computer Science, Springer, 2016, pp. 21–62.

[62] V. Bartsch, R. Machado, D. Merten, M. Rahn and F.-J. Pfreundt, "GASPI/GPI In-memory Checkpointing Library", in *Euro-Par 2017: Parallel Processing - 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28 - September 1, 2017, Proceedings* (F. F. Rivera, T. F. Pena and J. C. Cabaleiro, eds.), Lecture Notes in Computer Science, Springer, 2017, pp. 497–508.

[63] L. Dagum and R. Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming", *IEEE Comput. Sci. Eng.* **5** (1998), no. 1, pp. 46–55.

[64] O. Krzikalla, A. Rempke, A. Bleh, M. Wagner and T. Gerhold, "Spliss: A Sparse Linear System Solver for Transparent Integration of Emerging HPC Technologies into CFD Solvers and Applications", in *New Results in Numerical and Experimental Fluid Mechanics XIII* (A. Dillmann, G. Heller, E. Krämer and C. Wagner, eds.), Springer, 2021, pp. 635–645.

[65] A. Griewank, D. Juedes and J. Utke, "Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++", *ACM Trans. Math. Softw.* **22** (1996), no. 2, pp. 131–167.

[66] H. Jasak, "OpenFOAM: Open source CFD in research and industry", *Int. J. Nav. Archit. Ocean Eng.* **1** (2009), no. 2, pp. 89–94.

[67] D. Molinero, S. Galván, J. Pacheco and N. Herrera, "Multi GPU Implementation to Accelerate the CFD Simulation of a 3D Turbo-Machinery Benchmark Using the RapidCFD Library", in *Supercomputing* (M. Torres and J. Klapp, eds.), Springer, 2019, pp. 173–187.

[68] P. He, C. A. Mader, J. R. Martins and K. J. Maki, "Dafoam: An open-source adjoint framework for multidisciplinary design optimization with openfoam", *AIAA J.* **58** (2020), no. 3, pp. 1304–1319.

[69] M. Sagebaum and N. R. Gauger, *MeDiPack — Message Differentiation Package*, 2024. Online at https://scicomp.rptu.de/software/medi/ (accessed on October 31, 2025).

[70] F. Palacios, M. Colonno, A. Aranake, et al., "Stanford University Unstructured (SU^2): An open-source integrated computational environment for multi-physics simulation and design", in *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition 2013*, American Institute of Aeronautics and Astronautics, 2013, (60 pages).

[71] N. Beishuizen and U. Agrawal, *SU2GUI*, 2024. Online at https://github.com/su2code/su2gui (accessed on October 31, 2025).

[72] P. Singh and A. Manure, "Introduction to TensorFlow 2.0", in *Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python*, Apress, 2020, pp. 1–24.

[73] S. Imambi, K. B. Prakash and G. R. Kanagachidambaresan, "PyTorch", in *Programming with TensorFlow: Solution for Edge Computing Applications* (K. B. Prakash and G. R. Kanagachidambaresan, eds.), Springer, 2021, pp. 87–104.

[74] H. Topcuoglu, S. Hariri and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing", *IEEE Trans. Parallel Distrib. Syst.* **13** (2002), no. 3, pp. 260–274.

[75] L. F. Bittencourt, R. Sakellariou and E. R. M. Madeira, "DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm", in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, IEEE, 2010, pp. 27–34.

[76] H. Arabnejad and J. G. Barbosa, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table", *IEEE Trans. Parallel Distrib. Syst.* **25** (2014), no. 3, pp. 682–694.

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus. Mécanique
2025, Vol. 353, p. 1451-1475
https://doi.org/10.5802/crmeca.344

Research article

Towards an automated toolset for mesh adapted Navier–Stokes simulation of hypersonic vehicles

Mathieu Lugrin [*,a], **Baptiste Isnard** [*,a], **Clément Benazet** [b], **Bruno Maugars** [b] **and Cédric Content** [b]

[a] DAAA, ONERA, Institut Polytechnique de Paris, 92190 Meudon, France

[b] DAAA, ONERA, Institut Polytechnique de Paris, 92320 Châtillon, France

E-mail: mathieu.lugrin@onera.fr

Abstract. A fully automated CAD-to-post toolset for Navier–Stokes and Reynolds Averaged Navier–Stokes (RANS) simulations of high speed vehicles flow is presented. The toolset combines the vertex-centered SoNICS solver with anisotropic mesh adaptation based on the REFINE toolbox, eliminating manual meshing and enabling efficient, parallelized simulations from CAD input to converged solutions. The tool is validated on open cases representative of the complexity of the flow encountered around high-speed vehicles including reentry and airbreathing cruise vehicles. This includes complex three-dimensional forebody laminar simulations and axisymmetric triconic cases. For the internal aerodynamics part, given the lack of proper validation references, a direct numerical simulation of the internal conduit of a generic dual mode ramjet air intake is conducted and compared to legacy RANS simulation and mesh-adapted RANS simulation. A demonstration on a full scramjet-powered cruise vehicle then highlights the toolset's scalability and adaptability to industrial applications. The automated workflow reduces setup time and human error, making it a valuable asset for hypersonic vehicle design and optimization.

Keywords. Hypersonic, RANS, mesh adaptation.

Funding. This project was provided with computer and storage resources by GENCI thanks to the grant 2023-A0142A14106 on the SKL partition of the supercomputer Irene at TGCC. The development of the SoNICS CFD software used for this study was partly funded by the DGAC SONICE project.

Note. Article submitted by invitation.

Manuscript received 18 November 2025, accepted 4 December 2025.

1. Introduction

The design of high-speed vehicles heavily relies on Computational Fluid Dynamics (CFD), primarily through steady Navier–Stokes (NS) or Reynolds-Averaged Navier–Stokes (RANS) simulations. These simulations provide access to key performance indicators such as integrated data (e.g., lift to drag ratio, thrust, aerodynamic stability, etc.) or local quantities (heat fluxes, pressure load, etc.) that are then used to evaluate and refine the designs. Even with recent advances in

* Corresponding authors

terms of tools and solvers, setting up a CFD simulation from a CAD drawing of a system is still a time-intensive task that must be performed by a skilled engineer. While the use of unstructured meshes has significantly reduced the human time required to build medium to high quality meshes, meshing remains one of the main blocking points towards the use of CFD in automated toolchains. As the aerospace industry continues to push for more efficient design, the necessity for automated CAD to post-process workflows becomes increasingly evident. The manual CAD to mesh process, which is prone to human error and time-consuming iterations, obstructs the overall productivity of engineering teams. An integrated, automated meshing solution would significantly streamline the CFD pipeline, enabling faster design evaluation and optimization.

A possible solution for automatic CAD to post from the literature is the use of Immersed Boundary Method (IBM) [1]. The main advantage of this technique is that it does not require a body fitted mesh, and thus can use automatically generated Cartesian or octree meshes coupled with automatic masking of the area inside the solid. This method has been widely used for lattice-Boltzmann and steady or unsteady Navier–Stokes simulations, such as wall-modelled LES [2]. However, it usually suffers from multiple drawbacks. First, due to the absence of body fitted meshes, IBM simulations often rely on wall models, which have the common issue of lacking grid convergence [3] and exhibiting spurious behavior in separated regions [4]. This last drawback may be problematic for high-speed vehicle evaluation, particularly for cruise vehicles, where the air intake uses multiple shock-boundary layer interactions to slow down the flow and often reaches a fully separated state. Getting precise laminar baseflow using wall-model can also be challenging. Then, the use of body forces inherently leads to a non-conservative solution, which can again be a major problem for internal aerodynamics, as the mass imbalance may cause a wrong evaluation of the engine performance.

Another very promising solution to alleviate the manual meshing problems without the drawbacks of IBM is anisotropic mesh adaptation. It provides good convergence of the solution even with no physical intuition nor knowledge in the starting mesh, and do not suffer from conservativity or spurious behavior in separated regions, given that it remains resolved and body fitted. Anisotropic mesh adaptation is known for its high efficacy in the automatic generation of optimal computational meshes, first within the context of inviscid flow simulations [5–8] and more recently for RANS simulations [9,10]. At its heart, mesh adaptation [11–15] involves an iterative process: utilising data from a preceding simulation to automatically dictate the necessary mesh adjustments required to enhance either the global solution accuracy or a particular quantity of interest. This is achieved via solution-based sensors that analyse the current results and subsequently define modifications to the mesh. The goal of these prescribed modifications is to balance and optimise the element distribution — and, consequently, the computational resources — throughout the domain [16–18]. Regions characterised by significant gradients in the solution, or those deemed important for the computation of a specific output, are selectively and automatically refined to increase the local accuracy of the discretization. The validity and optimality of this methodology are underpinned by a formal mathematical analysis of the associated numerical error [19].

However, most toolchains relying on mesh adaptation are still not automated CAD-to-post workflows, as the tool often requires manually created surface and initial mesh of the model as well as the tagging of the boundary conditions. They are also often relatively inefficient, even if the mesh adaptation by itself allows for more efficient simulation due to the lower amount of points needed to converge a solution, as they chain tools through Input/Output (I/O, here used to refer to the writing and reading of the solution on a physical disk space) and different executables. Most of the tools are also not built to work in a parallel or distributed manner, which strongly limits performance and, worse, the maximal number of points that can be computed, as one may reach the memory limit of the computing nodes.

In this context, this paper presents a fully automated, parallel distributed CAD-to-post toolset for NS and RANS simulation based on anisotropic mesh adaptation, with a particular focus on the demonstration of its ability to capture the flow around high-speed vehicles.

While the tool has already been extensively validated during development using non-regression tests from the elsA solver, as well as open literature databases such as the NASA Turbulence Modeling Resource, conducting simulations of high-speed vehicles introduces new complexities that are specific to hypersonics and may not be covered by the usual CFD code validation. The tool must be able to handle highly compressible flow, including strong shocks, shock-shock, and shock-boundary layer interactions. This may also be challenging in terms of mesh adaptation, given that metric based adaptation will tend to refine regions with the strongest gradients (i.e. shocks) while overlooking other physically important regions of the flow, such as boundary and mixing layers.

To assess the ability of the tool to capture the flow around high-speed vehicles, it is tested on open cases representative of the complexity of the flow encountered in real-life applications, namely: a canonical reentry vehicle, a complex 3D forebody, and a supersonic internal air intake. Finally, a demonstration on a mock-up full cruise vehicle is conducted.

The article is organised as follows. First, the Navier–Stokes solver, the re-mesher, and the way they are linked together are presented. The manner in which the initial geometry, first mesh generation, and boundary condition tagging are handled is also highlighted, given that they are of great importance for the automation process. A new treatment of the metric used for adaptation and CAD handling is also presented. The tool is then validated on different open cases from the literature, representative of the complexity of real life applications.

The first case of interest is a canonical reentry vehicle that has been widely studied both numerically and experimentally [20,21]. It consists of an axisymmetric cone-cylinder-flare geometry, which offers numerous advantages. First, it allows for the comparison of the results and performances of the toolset on a bi-dimensional case against well established solvers from the literature. Even if geometrically simple, this case includes most of the complexity of the flows encountered around reentry vehicles.

The second and third cases of interest relate to airbreathing cruise vehicles. First, an external aerodynamic case is needed to validate that the tool can handle hypersonic forebodies, potentially with complex flow features such as curved shocks, the linked vortices, and high-Reynolds number laminar boundary layer. One main point of interest here will be the ability of the tool to capture accurate wall heat fluxes, which is both a challenging quantity to obtain numerically and one that is of paramount importance for the design of vehicles, as it will constrain the thermal protection system. Here, the BOLT case is chosen as a reference given that it provides geometrical and physical complexity close to that of real vehicles while being open and widely studied in the literature, with experimental (both ground [22,23] and flight [24,25]) and numerical results [26]. Specifically, a very challenging numerical reproduction of the highest laminar flight Reynolds number of the BOLT II campaign is conducted and compared to numerical results from [26].

The last validation case of interest relates to internal aerodynamics linked with ramjet propulsion. To get a properly functioning ramjet, one needs to slow down and compress the incoming supersonic flow to reach a subsonic state inside the engine. This can be done using both external and internal compression apparatus. Using a series of shock-waves, the flow is slowed down to feed the combustion chamber. To properly design the engine, one needs to have precise data about the flow that will enter the combustion chamber. Being able to easily and accurately compute such flows is thus mandatory to correctly predict vehicle trajectories and validate designs. For this case, the objective is to demonstrate that the tool can reliably reproduce the key internal flow features, such as corner vortices, shock boundary layer interaction (SBLI) and shock-train evolution. Even if some canonical cases of shock trains are available in the literature [27,28], they

are too far from applicative needs to be used in the scope of this paper. As such, a new validation case was built from scratch to reproduce all the flow features of interest. To establish a reference, a DNS of the benchmark configuration is performed, providing high-fidelity data against which results from the tool is compared. Given that the tool relies on turbulence modelling to capture such flow, a "legacy" RANS simulation (conducted with the elsA [29] well established solver) was conducted with the same models in order to discriminate discrepancies between the DNS and the RANS simulations caused by the turbulence modelling from those that may be caused by an ill behaviour of the toolset. Finally, once the ability of the tool to capture both complex internal and external flow has been proven, a demonstration on a full mock-up vehicle (with a modelled scramjet engine) is presented as a demonstration of the possible use of the tool.

2. Toolset description

The toolchain (presented in Figure 1) is built based on two main tools, namely the vertex-centered Navier–Stokes solver SoNICS from ONERA–Safran [30] briefly described hereafter and an anisotropic mesh adaptation toolbox adapted from the open-source toolbox REFINE developed by NASA [31]. Every step that will be described in the following is linked through memory APIs. This means that once the CAD file have be read by process, no I/O has to be conducted (except for post-treatment needs). Also, all following steps are executed in a parallel context, but parallel management is completely transparent to the user.

2.1. *CFD Solver*

SoNICS is a CFD software that solves Euler, Navier Stokes or RANS equations using either cell-centered or vertex-centered second-order finite volume schemes. It can perform multi-species reactive flows simulations. It has been designed with four major requirements: it must be easy to configure and must be able to take advantage of powerful hardware (CPU and GPU), which are all complex, distributed and heterogeneous [32]; it should handle a large variability in terms of Riemann solvers [33–35], flux limiters [36–38], fluid models (equations of state, viscosity laws, mixing laws...), turbulence modelling (RANS, LES, DES); it should be fully differentiable in order to allow modern optimization algorithms [39], input-output analysis [40–45], multi-disciplinary optimization [46,47], data assimilation [48–51], end-to-end machine learning [52]; and finally, it should provide APIs to help couple the tool with other solvers. A full description of the architecture in use for the SoNICS solver can be found in the companion paper by Lienhardt and Michel [53].

As mentioned above, the vertex-centered solver of SoNICS is used. The spatial discretization is based on the one proposed by A. Dervieux and C. Debiez [54] on simplex mesh (triangle/tetrahedron). The convective terms are solved by the finite volume method on the dual mesh composed of median cells. Fluxes are determined through the use of the AUPM approximate Riemann solver proposed by Chen et al. [34]. Second order space accuracy is achieved through MUSCL type reconstruction method [55]. The MUSCL procedure is applied as proposed in A. Dervieux and C. Debiez [54] or Alauzet and Frazza [56] to recover the so-called V4-scheme. A shock capturing similar to the one of [57] is also implemented. Viscous fluxes are computed with the conventional expression described as Cell Based Viscous scheme in Liu et al. [58].

Boundary conditions are weakly imposed by evaluating the flux at each facet at the boundaries of the domain. As in Alauzet and Frazza [56], each Dirichlet boundary (such as the velocities on a no-slip wall) are strongly enforced at each iteration, i.e. the DOF concerned by the boundary are removed from the whole CFD problem.

In order to reach the steady state, we consider an implicit backward Euler scheme with local time stepping, approximately solved by an efficient LU-SGS solver similar to the one proposed in [59]. For RANS simulation, the Spalart–Allmaras (SA) [60] model with the compressibility correction of [61] and the Quadratic Constitutive Relation (QCR) [62] are used.

2.2. *Anisotropic mesh adaptation on full tetrahedron meshes workflow*

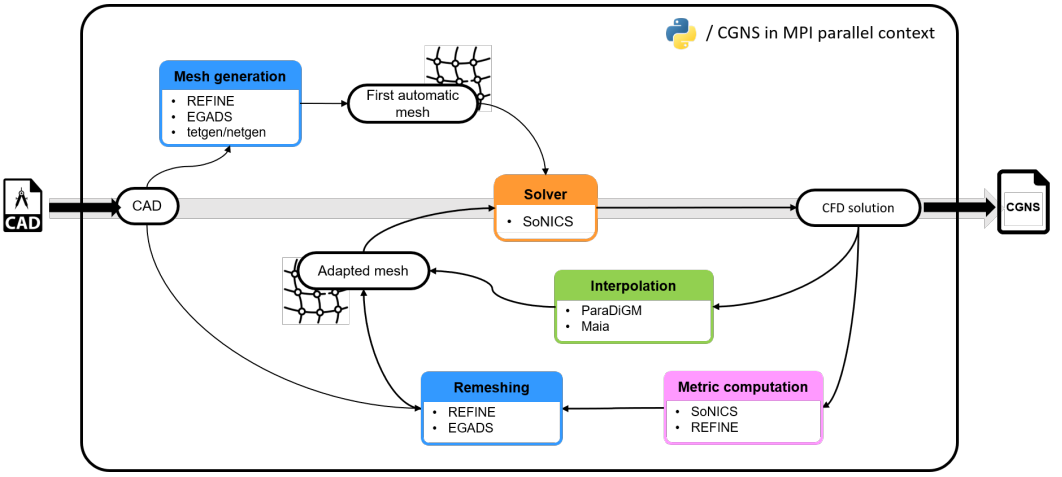


Figure 1. Diagram of the toolchain used for the automated computations.

The automated toolchain for mesh adaptation is presented in Figure 1. The input of the process is a CAD description of a closed shell representing the fluid volume to simulate. The first step is then to prove the existence of the mesh. For this purpose, REFINE provides an EGADS [63] wrapper to generate a surface mesh of the CAD shell. One can note that for the applications presented here, the CAD model is provided as a STEP file, which allows interaction with any CAD modeling software; however, a direct link with ESP [64] could also be utilized to take advantage of its differentiation. From the STEP input, EGADS tessellates the CAD edges and surfaces to build a closed surface mesh. Tesselation provides meshes whose elements have poor quality for a volume mesher, especially given that each CAD patch is meshed independently. As such, the tesselation process results in an overall low-quality surface mesh. In order to improve its quality before providing it to the tetrahedral mesher, REFINE performs surface mesh adaptation driven by a curvature-based metric that is evaluated thanks to the CAD parametrisation at each mesh node. Several passes can be performed with various parameters (minimal and maximal cell size, mesh complexity, size gradation) in order to obtain a decent surface mesh before volume mesh generation. Once done, a tetrahedral mesh generator (here tetgen [65] or netgen [66]) is used to mesh the interior of the fluid volume. This first mesh generation is illustrated in Figure 2.

The mesh structure — including coordinates, connectivity, and CAD information — is converted into a Python/CGNS structure [67] (CGNS tree), which serves as the entry point for the SoNICS solver. The CGNS tree data is natively distributed over MPI processes following Maia's parallel rules for CGNS trees (see [68,69]). It will remain distributed for the rest of the workflow.

In industrial configurations, the number of ridge and surface patches can be exceedingly large, which implies some extra work for the user to retrieve link between CAD patches and the CFD boundary conditions (BCs). In order to relieve the user of this step, if CAD patches are

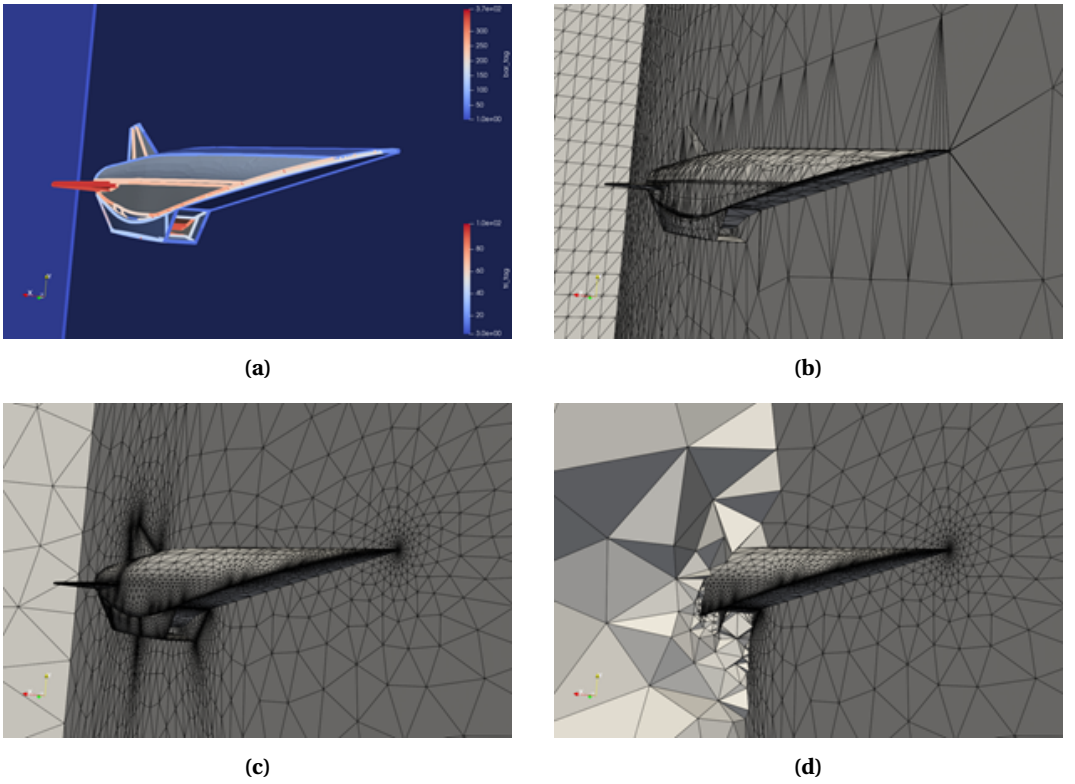


Figure 2. Automated first mesh generation process illustrated by: (a) CAD model; (b) the EGADS tesselation of the CAD model; (c) the surface mesh after remeshing by REFINE; (d) volume mesh created by tetgen.

tagged with names using the advanced face option of STEP format, the toolchain preserves these names during the mesh generation process, so boundaries can easily be retrieved for the solver boundary condition tagging. Patches from the same families can also be concatenated together in a single CGNS node to ease I/O and tree handling.

The Python/CGNS tree is then prepared by adding a family of boundary conditions (BCs) (automatically, thanks to the preservation of patch names), the reference state, and the solver options. Steady compressible Navier–Stokes or RANS simulations are then conducted using the vertex-centered solver of SoNICS described here-before. Once a solution is found, the computation of the mesh adaptation metric is performed directly in the CFD solver and is fed to the remesher. In this work, the metric is computed as the absolute value of the Hessian of the Mach number (albeit the user can decide to use any field). A pre-processing is applied by REFINE to compute the error estimate for the desired p-norm and complexity. Then an anisotropic size gradation algorithm is used to smooth the metric. REFINE then performs a parallel remeshing of the surface and volume meshes. The geometric description of the fluid domain is preserved with high fidelity using EGADS's CAD projection when inserting or moving boundary mesh vertices. The current flow solution is finally interpolated on the new adapted mesh using the open-source ParaDiGM library [70] (see also [71]), which is available through the Python/CGNS API of Maia. The interpolated solution is used as an initial state and the simulation is run again. The process is looped, potentially with an increase in complexity depending on the steps, until a proper solution is converged. Convergence indicators are case-dependent and are to be included in the Python

script directly by the person skilled in the art. The same goes for complexity jumps, gradation, and most of the CFD solver parameters which are also application-dependent.

The whole procedure that is described above is performed by a single Python script that takes as inputs the STEP file and the run conditions. All pre/co/post-processing, partitioning and distribution are performed using the open-source Maia library.

2.3. *Convergence*

Given that the first generated mesh is very rough and does not include any physical intuition, the first NS or RANS computation at the beginning of the loop can be numerically challenging. The usual way to start the mesh convergence process is to run the solver computation over a few pseudo-time iterations to avoid solver crashes. The number of iterations can then be slowly increased over remeshing passes to gradually converge the proper solution. A trick allows to accelerate this solution convergence process: the Mach Hessian used as a metric for feature-based remeshing is a good criterion for mesh convergence of Navier–Stokes equations resolution [72]. However, if used as is, it can lead to slow mesh convergence because it is not very efficient at quickly capturing the boundary layer. Especially for flows including shocks, in the first steps where meshes have poor discretization, the chosen metric highlights the very strong gradients in compression regions compared to viscous regions. Frazza [9] proposed to boost the Hessian near viscous walls in order to add more points into the boundary layer, which will help the process converge quickly to a steady flow solution. For this purpose, metric diagonal is boosted by a factor in the wall normal direction.

3. Reentry vehicles aerodynamics

As stated in the introduction, the first type of high-speed vehicles of interest are the reentry vehicles. To assess the ability of the solver to treat those cases, the axisymmetric cone-cylinder-flare CCF12 geometry, which was designed as a canonical and open case to study the flow features found in atmospheric reentry vehicles is chosen as a benchmark. It includes a simple developing hypersonic laminar boundary layer followed by more complex features such as boundary layer interaction with expansion fans, and most importantly, separation induced by the shock-boundary layer interaction. The numerical results of Caillaud et al. [20] (see Table 1) are used as a reference given that they provide high-order solutions cross-validated between two well established solvers from ONERA [73] and NASA [74]. For this case, the axisymmetric solver of SoNICS is used alongside the 2D remesher of REFINE, in this case the mesh is only composed of triangles.

In that specific 2D axisymmetric case, the CAD file is generated from the analytical description of the body using GMSH and exported as a STEP file. An example of an adapted mesh on that case, alongside a Mach number contour, is presented in Figure 3.

Taking the low Reynolds case as an example, the toolset is run for 121 seconds on 4 cores of an Intel Cascade Lake CPU, allowing us to conduct 6 mesh adaptations and steady computations

Table 1. Flow conditions from [20] used in this study.

$\mathrm{Re}_m \times 10^6$ (m^{-1})	R_n (mm)	u_∞ (m/s)	ρ_∞ ($\mathrm{kg/m}^3$)	T_∞ (K)	T_0 (K)	T_{wall} (K)	P_0 (kPa)
2.229	0.1	907.92	0.00920	56.977	467.21	300	237.50
4.614	0.1	990.41	0.02128	67.800	555.96	300	653.90

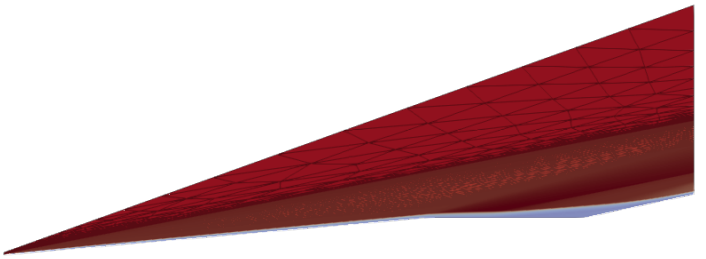


Figure 3. Adapted mesh and Mach number contour for the low Reynolds number case of [20].

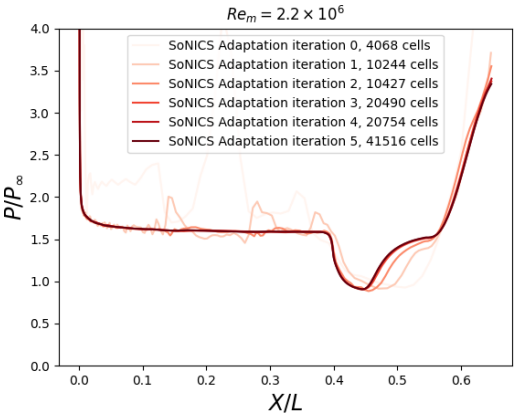


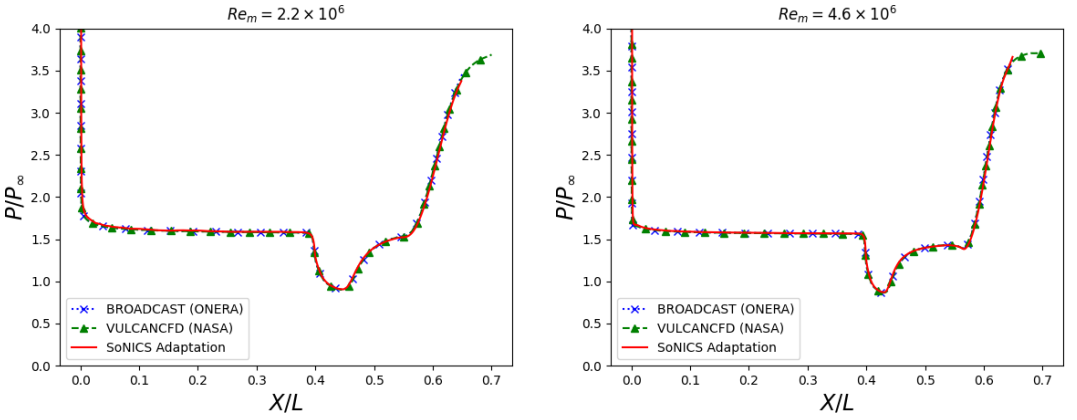
Figure 4. Evolution of the wall pressure distribution on the CCF12 at low Reynolds number at each iteration showing the convergence path.

until complete convergence is reached, the evolution of the wall pressure distribution for each step is presented in Figure 4.

The wall pressure distribution for two different Reynolds numbers is presented in Figure 5 alongside results from two high-order solvers. Given the sensitivity of the strongly separated shock boundary layer interaction to both the resolution of the boundary layer gradients and the associated separation and reattachment shock, obtaining the correct pressure signature guarantees that these features are accurately captured in the solution. This result validates the good behaviour of the toolset for hypersonic axisymmetric configurations.

4. Cruise vehicles aerodynamics

The second type of vehicle of interest is air-breathing cruise vehicles, which, in addition to external aerodynamics, involve complex internal flow. This validation is split into two parts: first, a 3D forebody case is studied; then, a canonical internal aerodynamic reference case is built and used as a reference to validate the toolset.



(a) Low Reynolds case **(b)** Intermediate Reynolds number case

Figure 5. Wall pressure distribution for the two different Reynolds number alongside reference data from [20].

4.1. *Forebody*

The first subcase of interest for cruise vehicles concerns forebodies, which are often complex 3D geometries on which the main points of interest will be drag, lift and heat-flux prediction. The BOLT case is chosen as it provides plenty of open data (including flight) and its 3D geometry is representative of the complexity of real designs. It also displays an intricate baseflow structure with multiple vortices, which is challenging to capture and is thus a good benchmark. The chosen condition for the computation is a challenging high Reynolds case with both CFD and sparse flight data coming from [26]. This case is at a flight altitude where laminar-turbulent transition starts to appear, which means that it is the highest Reynolds number for which the flow remains laminar; the full flow conditions are presented in Table 2. Given the high Mach number and the relatively low altitude, this case is fully representative of a scale one hypersonic cruise vehicle forebody in flight. This case would also serve future validation of transition prediction tools integrated to the solver.

The case is run on a quarter of the full scale flight BOLT II model using symmetry planes. The computation is conducted on 2400 Intel Cascade Lake cores for a single job of 15 hours, allowing for 8 adaptation/computation steps and achieving a final mesh of 61 million nodes in order to reach proper convergence of the wall heat fluxes. This can be compared to the 110M cells used in the study of [26], keeping in mind that the solution on the structured grid is computed using a sixth order central finite difference scheme, while the mesh adapted solution uses a second order method. This gives insight on the ability of mesh adaptation to strongly reduce the required number of DOF to achieve the same convergence. This could be even more drastic when using goal-orientated adaptation (on the parietal heat-flux) [75], given that the convergence of

Table 2. Flow conditions from [26] used for the BOLT forebody reference, data coming from the BOLT-II flight.

Time (s)	$\mathrm{Re}_\infty \times 10^6$ (m^{-1})	Altitude (m)	u_∞ (m/s)	ρ_∞ (kg/m^3)	T_∞ (K)	Mach	T_{wall} (K)
401.9	6.0914	23908	1768.5	0.048302	213.15	6.0424	390

wall heat-fluxes requires a lot of degrees of freedom, especially with meshes composed only of tetrahedron and using metric based adaptation.

An illustration of the geometry, alongside the wall heat-flux distribution and a Mach number field at the outlet of the domain, is presented in Figure 6. It highlights the complex nature of the flowfield with multiple vortical structures appearing, especially in the center region. Quantitative comparisons of heat-fluxes at two different longitudinal positions with the CFD solution of [26] are presented in Figure 7. Overall, excellent agreement is found between the two simulations. Results from the mesh adaptation display a relatively good smoothness given that they come from anisotropic tetrahedron only meshes. Some discrepancies are still visible in the peaks linked with local vortices, mostly in terms of amplitude of the fluctuation. Figure 7(a) presents the most upstream slice, from left to right, the relatively low heat-flux region is linked with the upwash of the centerline vortices (caused by the 3D nature of the geometry and thus of the bow shock which leads to the creation of vorticity), those vortices are clearly visible in Figure 6. This region is then followed by a first peak linked with the downwash of those vortices and then a long plateau of relatively high heat flux linked with the thin boundary layer in the outboard region. In all those regions, the two computations are in almost perfect agreement.

The origin of these discrepancies is still unknown; they may be linked to differences in the mesh resolution; some vortices may be under-resolved in either simulation. Another possible origin could be the growth of vortices linked to numerical errors from the time integrators, as the baseflow is extremely unstable; small numerical errors could lead to the growth of structures that are not part of the baseflow. Future work on that case will require additional simulations from other meshes and/or solvers to assess the origin of these discrepancies. Again, a Newton method could be used once the Jacobian operator is available to compute a fixed point of the equations.

Overall, the tool appears to be fully capable of computing a very complex baseflow such as the one encountered on the BOLT forebody. The key point to highlight here is that the user time needed to set up such a computation is minimal (of the order of minutes) with no effort whatsoever put into meshing, which strongly contrasts with other studies on that case.

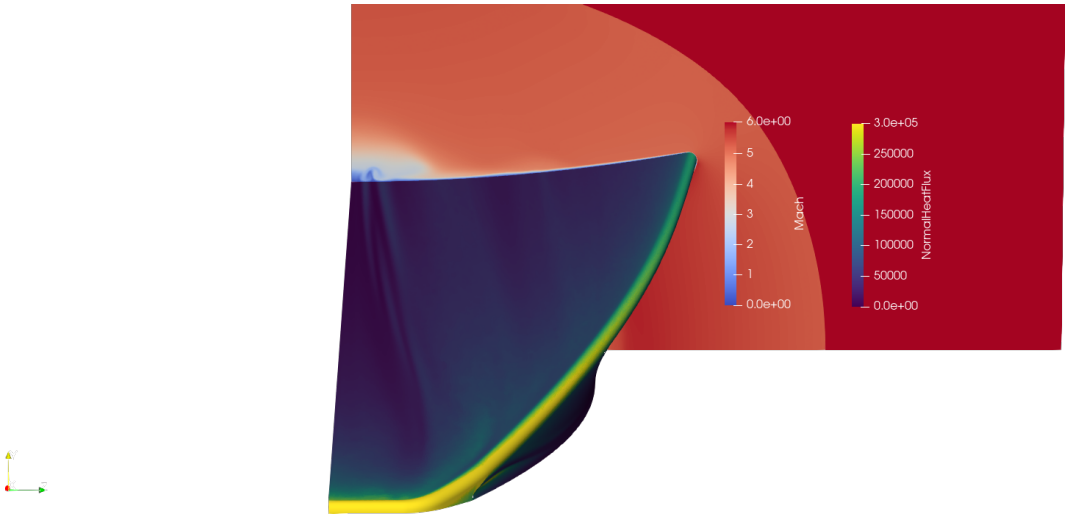
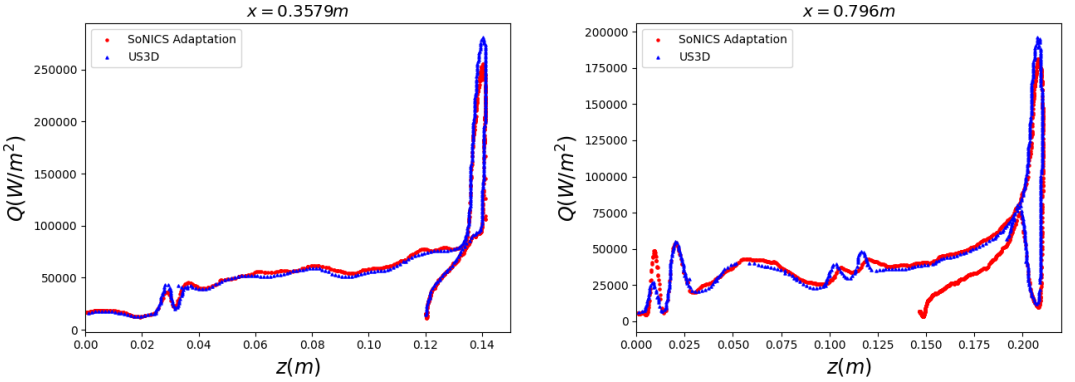


Figure 6. BOLT geometry and simulation results (wall heat flux and Mach number field at the outlet of the simulation domain) for the descent case of [26].



(a) **(b)**

Figure 7. Spanwise heat-fluxes distribution at two longitudinal sections of the BOLT forebody alongside data from the US3D [76] computation of [26].

4.2. *Internal aerodynamics case*

As explained in the introduction, given the lack of open cases representative of applied vehicle technologies, a reference case is built to validate the toolset on hypersonic internal aerodynamics. The motivation for the geometry and flow conditions choices as well as the setup for the reference simulation is presented in this section, followed by comparison between reference, legacy RANS and automatic toolset results.

4.2.1. *Geometry and flow conditions*

The geometry under study is designed to be representative of the complexity of the internal flow of a ramjet powered vehicle. As such, it should include an internal compression region, followed by a throat, and then a slowly diverging section in which a shock train will appear. A back pressure at the outlet of the diverging section is necessary to simulate the engine. To facilitate the sharing and accurate reproduction by other researchers, the geometry must be canonical and analytically describable. This led to the geometry described in Appendix A. The dimensions of the geometry, while arbitrary, are chosen such that reproductions in conventional supersonic wind tunnels are facilitated. The flow conditions are chosen to match a reasonable value for most supersonic wind tunnels to allow for further cross comparison.

The Reynolds number is chosen as high as possible, while still allowing for a highly resolved computation within the given computing allocation. The inlet Mach number is set to 3, the stagnation temperature is low (300 K) to facilitate experimental reproduction, and the static pressure of the inlet is chosen to adjust the Reynolds number and thus the computational cost. The complete conditions, as well as the reference values used for the computations, are presented in Table 3.

Table 3. Reference conditions for the computations.

M_∞	P_∞	T_i	α	β	T_{wall}	Re_L	L
3	4000 Pa	300 K	0°	0°	300 K	4.029×10^6	0.4 m

The boundary conditions are supersonic inlet and outlet, and isothermal viscous wall at 300 K. The turbulence generation technique is described in Appendix C. The chosen conditions lead

to the approximate Reynolds number presented in Table 4, θ and δ_1 being the boundary layer momentum and displacement thickness, respectively.

Table 4. Approximate boundary layer thickness and boundary layer based Reynolds number in the central plane $z = 0$.

	Re_θ	δ_1
Before first ramp ($x = 0$)	1500	0.00077 m
At bottom wall separation point (see Figure 9)	6875	0.00169 m

4.2.2. *Reference numerical simulation setup*

In order to conduct the DNS and the legacy RANS simulations, a structured mesh is created. The characteristic sizes of the meshes are presented in Table 5, the wall units sizing are computed from an initial SA QCR RANS computation, and the maxima are given compared to the local wall unit of the closest wall, ensuring good resolution in the entire domain. Stream-wise cell sizes are adapted using linear laws, whereas wall-normal ones are using a collection of geometrical laws. The final DNS mesh contains more than 3.5 billion points without ghost cells. The RANS mesh uses the same construction, but with much less resolution.

Table 5. Mesh characteristics.

	n_x	n_y	n_z	n_{tot}	$\Delta x^+_{\mathrm{max}}$	$\Delta y, z^+_{\mathrm{max}}$	$\Delta y, z^+_{\mathrm{wall}}$
DNS	9807	605	605	3.59×10^9	< 12	< 10	< 1.4
RANS	442	195	195	16.8×10^6	< 300	< 100	< 1.4

The DNS (see Figure 8) is performed using the open-source HPC oriented FAST compressible Navier–Stokes solver from ONERA [77]. The solver uses a finite volume approach on multi-block structured mesh using hybrid MPI/OpenMP parallel programming, cache blocking, and vectorization to improve performance. Inviscid fluxes are solved using an extended version of the scheme described in [78], details about the spatial scheme are given in Appendix B. The viscous fluxes are computed using a second-order accurate centered scheme. Time integration is performed via an explicit low-storage third-order 3-step Runge–Kutta scheme; the time step is set to $\Delta t = 3 \times 10^{-9}$ s to ensure a CFL number lower than 0.5 in the whole domain. The molecular viscosity is computed using Sutherland's law. The computation was run using a hybrid MPI/OpenMP parallelism on 64 computing nodes (3072 cores), using 1024 MPI processes and 3 OpenMP threads per process. The mean flow is averaged over 2×10^6 iterations, which corresponds to a physical time of 6 ms.

The legacy RANS simulations are performed using the elsA software (Safran–ONERA property [29]) on the structured mesh presented before (see Table 5). The inviscid fluxes are computed using the AUSM+up of Liou [79] and an unlimited third-order MUSCL reconstruction. The viscous fluxes are solved using a centered scheme on five points, and the time integration to find the steady solution is done implicitly with a local time-stepping approach. In order to obtain the correct solution, the simulation is initialized with a subsonic region in a subpart of the diverging section and free-stream values elsewhere, the same initialisation strategy is used for the mesh adapted simulation.

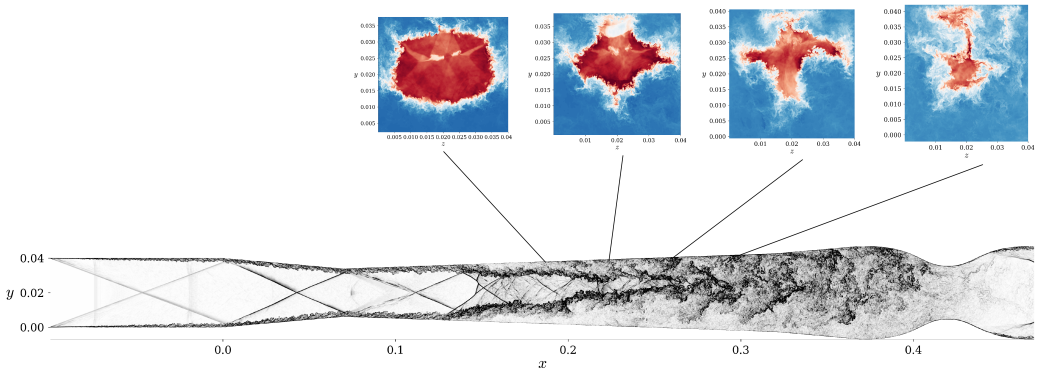


Figure 8. Illustration of the simulation thanks to instantaneous numerical pseudo-schlieren in a constant z-plane ($z = 0.01$) of the configuration and instantaneous stagnation pressure contour in various x-plane.

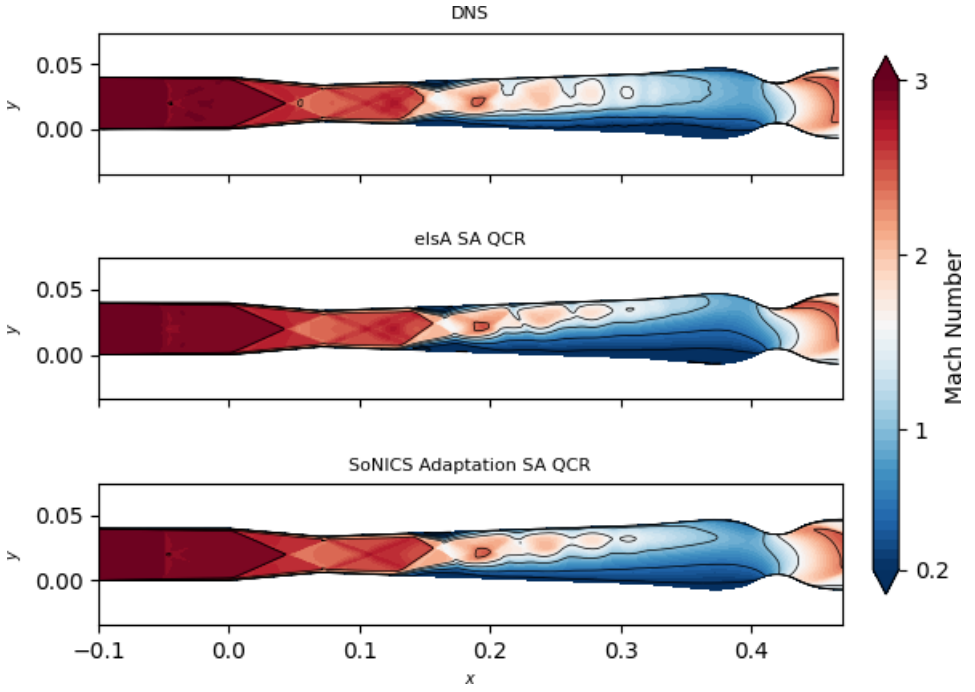


Figure 9. Mach number distribution in the central z-plane. The DNS results correspond to time-averaged flow.

4.2.3. *Results*

For this case, given the slow time convergence of internal flow, the tool is run for 33 adaptation steps with a varying number of time iterations, to reach a final mesh of 4 million nodes (to be compared to the 17 million cells of the legacy setup). The time-averaged Mach number fields (in the central plane) for the DNS and both RANS simulations are presented in Figure 9. The

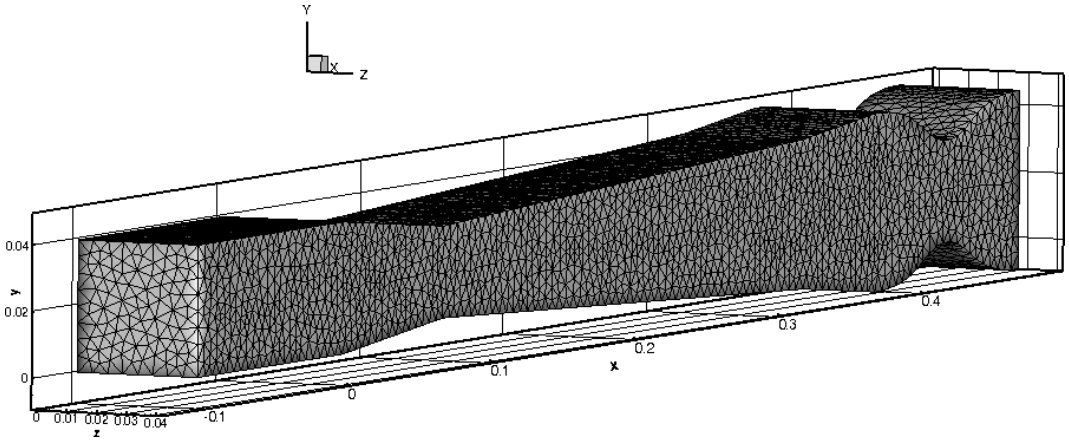


Figure 10. Starting mesh for the mesh adapted RANS simulation.

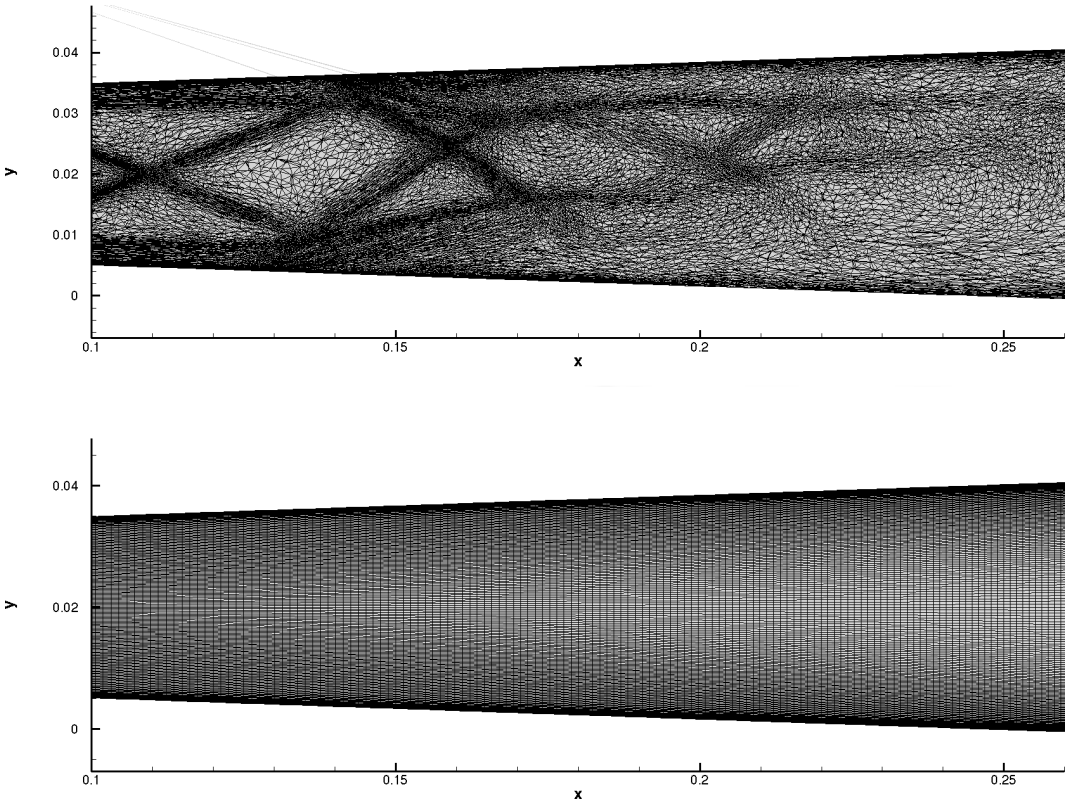


Figure 11. Meshes from the adapted (top) and elsA (bottom) SA QCR simulations in the central plane for the shock-train region.

flow displays all the main expected features: first, a boundary layer develops, then encounters a first shock wave as it reaches the compression ramp, followed by an expansion at the entrance of the diffuser. The flow ends up separating via a SBLI and is slowed down by a shock train to finally reach a purely subsonic state. Both RANS simulations display results that are surprisingly

close to the DNS reference, given the complexity of the flow (one can note here that the use of the QCR strongly improves the quality of the RANS solution, as described in Appendix C). From an engineering standpoint, both methods are acceptable for vehicle design. This validates the good behavior of the toolset for turbulent internal aerodynamics. Upon closer examination of the solutions, it is evident that the elsA results show a shock train located closer to the top wall than in the mesh-adapted simulation. Consequently, the latter is in better agreement with the DNS reference. This may be due to the traditional meshing procedure used for the standard simulation, where the mesh is highly refined near the walls to properly capture the boundary layer, leading to a mesh-induced displacement of the shock train. As the refinement of the mesh-adapted simulation is fully feature-driven, no such best practice is used (see the initial mesh presented in Figure 10), which leads to a better resolution of the mesh further away from the wall. The difference between both final meshes in the central plane for the shock train region is displayed in Figure 11. The adapted mesh clearly displays a refined region further away from the wall in addition to the near-wall refinement for the boundary layer, which may allow a better capture of the physics of the flow.

5. Demonstration on a full vehicle

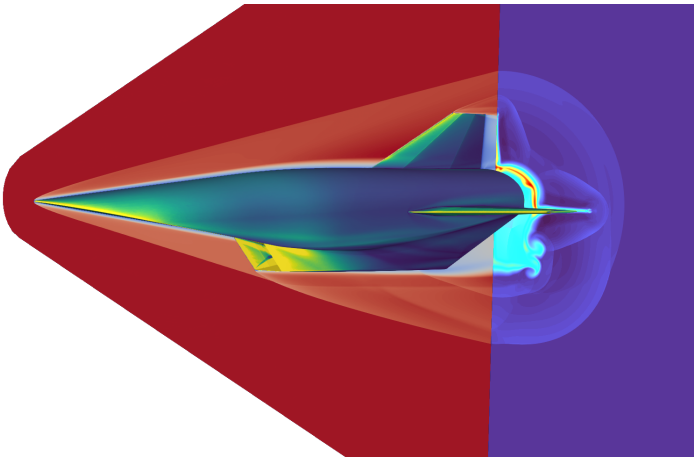


Figure 12. Skin-friction, Mach number slice in the central plane and temperature slice at the outlet illustrating the external aerodynamics of the full vehicle.

Now that the ability of the toolset to capture both complex internal and external high speed aerodynamic flowfield has been documented, a demonstration on a full mock-up vehicle can be conducted. The case consists of a high Mach number cruise of a scramjet powered vehicle. The combustion chamber is not computed but replaced by a 0D model extending from the air intake throat to the nozzle entrance. This simulation only serves as a demonstration of how we can leverage the ability of the toolset to handle complex geometry at no additional cost. Setup time for such a simulation requires only few minutes of human work. The toolset is then run on 600 Intel Cascade Lake cores for a single job of 15 hours, which allowed conducting 12 adaptation/simulation steps to reach convergence of the aero-propulsive balance, the final mesh has 35 million nodes. Samples of results are presented in Figures 12 and 13. Although quantitative data cannot be presented here, the objective is to demonstrate the tool's ability to manage complex geometries (e.g., control surfaces and bleed systems). A key advantage is that

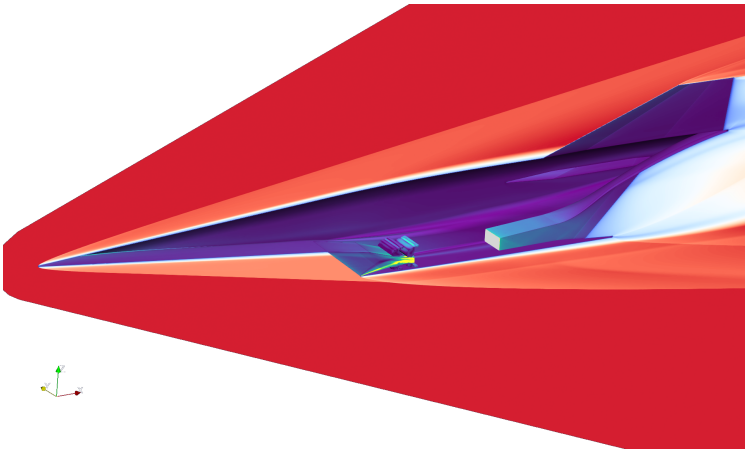


Figure 13. Skin pressure distribution and Mach number slice in the central plane illustrating the internal aerodynamics of the full vehicle.

geometric variability is fully transparent to the user, for example, the same simulation can be performed with different or additional control surfaces simply by modifying the input CAD file.

6. Conclusion

This study presents the development and validation of a fully automated, CAD-to-post toolset designed for Navier–Stokes and Reynolds-Averaged Navier–Stokes (RANS) simulations of complex hypersonic vehicles. The toolset integrates a 2D, 2D-axi and 3D vertex-centered finite volume Navier–Stokes solver, SoNICS, with an anisotropic mesh adaptation framework based on the REFINE toolbox. By leveraging automated CAD handling, surface and volume meshing, and metric-based remeshing, the toolset eliminates the need for manual intervention in the meshing process, significantly reducing setup time and human error. The workflow is entirely parallelized and managed through a Python/CGNS interface, ensuring scalability and efficiency on modern high-performance computing architectures. The toolset was rigorously validated using a series of open benchmark cases representative of the complex flow physics encountered in hypersonic applications. First, the axisymmetric cone-cylinder-flare geometry, a canonical reentry vehicle configuration, demonstrated the toolset's ability to accurately capture shock-boundary layer interactions and flow separation, with results closely matching high-order reference solutions. Second, the BOLT forebody case, characterized by its three-dimensional geometry and intricate vortical structures, highlighted the toolset's robustness in predicting wall heat fluxes and complex laminar boundary layers at high Reynolds numbers. The comparison with experimental and numerical data from the literature confirmed the toolset's reliability in resolving challenging external aerodynamic features. For internal aerodynamics, a novel validation case was designed to replicate the flow conditions within a ramjet-powered vehicle's air intake. This case included a compression ramp, diffuser, and shock-train formation, all of which are critical for engine performance evaluation. The toolset's results were benchmarked against a Direct Numerical Simulation (DNS) and a legacy RANS simulation using the elsA solver. The comparison revealed that the mesh-adapted RANS simulations, particularly when using the Quadratic Constitutive Relation (QCR) turbulence model, provided results remarkably close to the DNS reference. This underscores the toolset's capability to capture key flow features such as shock-boundary layer interactions, corner vortices, and shock-train evolution, even in highly turbulent and separated flow

regimes. Finally, a demonstration on a full mock-up of a scramjet-powered cruise vehicle illustrated the toolset's scalability and adaptability to complex geometries. The simulation, which included both external and internal aerodynamics, was set up with minimal human effort and efficiently converged on high-performance computing resources. This showcases the toolset's potential for industrial applications, where rapid design iterations and high-fidelity simulations are essential. The results presented in this study validate the toolset's ability to handle the full spectrum of hypersonic flow complexities, from external aerodynamics to internal propulsion systems. The automated workflow, combined with anisotropic mesh adaptation, ensures both accuracy and computational efficiency, making it a valuable asset for future hypersonic vehicle design and optimization. Future work will focus on extending the toolset's capabilities to include moving bodies (through CAD manipulation), hybrid meshes, multi-species flows, advanced turbulence modeling, and GPU-accelerated remeshing to further enhance its performance and applicability. Additionally, the integration of adjoint-based optimization and data assimilation techniques will enable more sophisticated design and analysis workflows, paving the way for next-generation hypersonic vehicle development.

Acknowledgments

The manuscript was written through contributions of all authors.

Philippe Duveau and Pierre Grenson are acknowledged for the discussion about internal aerodynamics and help to define the reference geometry, as well as Sebastien Deck, Nicolas Renard, Jaime Vacquero, Julien Husson and Lucas Manueco for the discussions on turbulence modelling and high-fidelity simulations. The authors would like to thank Bastien Andrieu for his constant effort in Paradigm improvement and for all the discussions on geometry, mesh and partitioning. Last, the authors would like to thank Ivan Mary for the help and support regarding the use of the FastS solver and numerical scheme implementation.

Declaration of interests

The authors do not work for, advise, own shares in, or receive funds from any organization that could benefit from this article, and have declared no affiliations other than their research organizations.

Appendix A. Geometry description for the internal aerodynamics case

This appendix describes the geometry that was designed to build the internal aerodynamics reference case.

The cross plane is top/bottom symmetrical (see Figure 14) consisting of first a 10 cm long flat plate to allow for the formation of a proper turbulent boundary layer (the method used to ensure a turbulent inflow is presented in Appendix C). Then, the internal compression is caused by a 7 cm long, 5° compression ramp. This ramp is followed by a long $-2°$ diverging section that spans over 25 cm. This section is representative of the diffuser in a ramjet air intake, and its angle was selected to prevent boundary layer thickening from further restricting the inviscid flow area. Finally, the flow enters a fast-diverging section with a $-5°$ angle, extending over 5 cm. To ensure a well-posed boundary condition, back pressure is generated by incorporating a choked converging-diverging nozzle downstream of the diverging section. This configuration enables the use of a supersonic outflow boundary condition. The nozzle shape is defined using a Hermitian spline. An analytical (or discretized) description is available upon request. The throat closure has

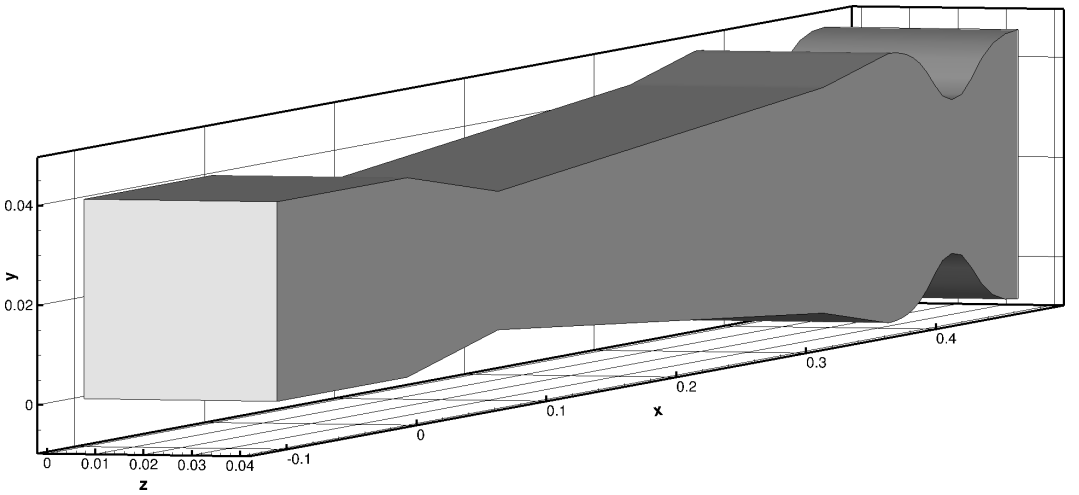


Figure 14. 3D representation of the geometry under study.

been chosen so that the isolator behaviour is close to critical for the flow conditions under study (terminal compression near the start of the diffuser). However, being too close to the critical point could have led to the unstart of the air intake either during DNS transient or when using some turbulence models. To alleviate this problem, the final condition under study is what could be considered an off-design condition, where the air intake does not provide the maximal efficiency. The total height of the inlet plane in 4 cm. The 3D geometry is then created by extruding that plane over 4 cm, creating a square shape at the inlet of the domain.

Appendix B. Numerical method for the DNS

The approximate Riemann solver is the simplified AUSM(P):

$$F_c^{i+\frac{1}{2}} = U_1 \frac{\mathbf{q}_R + \mathbf{q}_L}{2} - \Phi U_{\text{dis}} \frac{\mathbf{q}_R - \mathbf{q}_L}{2} + P, \tag{1}$$

U_1 and U_{dis} being the interface fluid velocity and the dissipation velocity similar to the original formulation of the scheme [78], $\mathbf{q}$ is the conservative state variable vector and the subscript $\cdot_L$ (resp. $\cdot_R$) corresponds to left (resp. right) state computed thanks to a MUSCL reconstruction, Φ is the dissipation sensor which is discussed hereafter.

In order to limit the dissipation, Mary [78] has proposed to identify regions where cell-to-cell oscillations vanish in order to fine tune a 2nd order low dissipation scheme. The proposed oscillations sensor reads:

$$\Delta_i^k = \begin{cases} -1 & \text{if } (\phi_{i+1}^k - \phi_i^k)(\phi_i^k - \phi_{i-1}^k) < 0, \\ 1 & \text{otherwise,} \end{cases} \tag{2}$$

with ϕ_i^k the k-th component of the state vector of primitive variables at point i. Then the dissipation sensor is defined as:

$$\Phi^i = \max_k \begin{cases} 1 & \text{if } \sum_{l=i}^{i+1} \Delta_l^k \leq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

If the solution is fully monotonous in the stencil, the scheme behaves as a fully centered scheme. If the solution is nonmonotonous, the dissipation is fully activated.

In this work, we proposed to extend the methodology to fifth-order MUSCL reconstruction. Due to the extent of the stencil the upwinding function of the scheme must be reworked. The smoothness of the solution is still evaluated by the number of slope changes in the stencil. If there are one, two, or more, the effective dissipation is set to an increased amount of the standard upwind scheme:

$$\Phi^i = \max_k \begin{cases} 1 & \text{if } \sum_{l=i-1}^{i+2} \Delta_l^k < 0, \\ \alpha_2 & \text{if } \sum_{l=i-1}^{i+2} \Delta_l^k = 0, \\ \alpha_1 & \text{if } \sum_{l=i-1}^{i+2} \Delta_l^k = 2, \\ \alpha_0 & \text{if } \sum_{l=i-1}^{i+2} \Delta_l^k = 4, \end{cases} \tag{4}$$

with $0 < \alpha_i \leq 1$ and $\alpha_0 < \alpha_1 < \alpha_2$. This approach strongly limits the dissipation of the scheme in regions where the solution is smooth, while preserving the stability of the scheme in non-smooth regions.

For supersonic and hypersonic flows, in order to ensure the robustness of the simulation while preserving the high resolvability properties of the scheme in the smooth flow regions, the shock capturing based on the work of [57] is used. The discontinuity sensor reads:

$$\kappa_i = \frac{10}{2}\left[1 - \tanh\left(2.5 + 10\frac{\delta x_i}{c}\nabla\cdot\mathbf{u}\right)\right] \times \frac{(\nabla\cdot\mathbf{u})^2}{(\nabla\cdot\mathbf{u})^2 + |\nabla\times\mathbf{u}|^2 + \epsilon} \times \left|\frac{p_{i+1} - 2p_i + p_{i-1}}{p_{i+1} + 2p_i + p_{i-1}}\right|, \tag{5}$$

with δx_i the size of the cell in the direction of interest, c the speed of sound, p_i the pressure at the point i and $\mathbf{u}$ the velocity vector. The sensor is used to switch from fifth to first order MUSCL state reconstruction, as following:

$$\mathbf{q}_{R,L} = (1-\kappa)\mathbf{q}_{R,L}^{o5} + \kappa\mathbf{q}_{R,L}^{o1}. \tag{6}$$

In strong shocks, the reconstruction tends towards first order, while elsewhere it stays at fifth order, ensuring a proper stability of the numerical method in discontinuities. As the sensor is directional, a different coefficient is applied to the reconstruction in each direction. An instantaneous map of κ for the i-direction in a constant z-plane ($z = 0.01$) of the DNS is presented in Figure 15, showing that the first order reconstruction use is confined to shock-wave and does not reduce the order of the scheme in turbulence nor boundary layers.

For instance, the numerical scheme depicted above with tuning parameters $\alpha_0 = 0.05$, $\alpha_1 = 0.4$, $\alpha_2 = 0.6$ has been successfully used in [80].

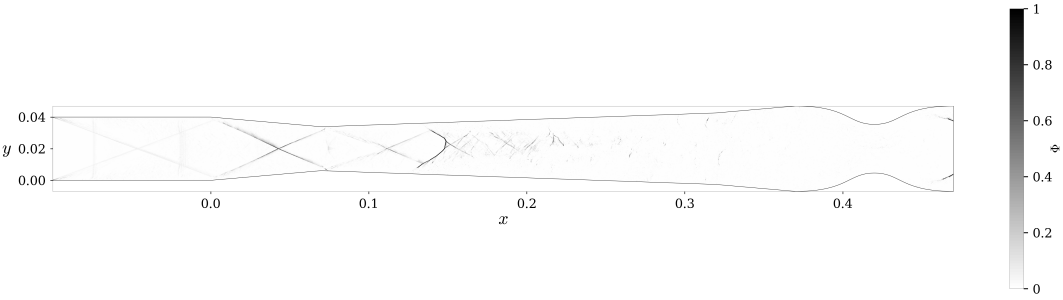


Figure 15. Instantaneous map of κ in the i-direction in a constant z-plane ($z = 0.01$) of the configuration.

Appendix C. Turbulent inflow generation for the DNS

One important aspect for such DNS is the need to inject turbulence at the inlet of the domain. In the present case it was decided not to use a standard turbulence generation method (such as Synthetic Eddy Method of Digital Inflow Filtering, which may both introduce nonphysical acoustic fluctuations which might be detrimental in a confined environment), instead a method inspired from recent advances in natural transition simulation [81,82] is used. To create a quick, and low noise transition, streaks are first generated by alternating viscous and non-viscous wall boundary conditions patches at the inlet of the domain. The wavenumber of the created streaks is chosen to be as small as possible, the lower limit being linked to the resolution of the scheme. The patches are 4 cells wide and are repeated every 12 cells. The length of the patches is set to 40 cells. The created streaks represent a very efficient support for transitional instabilities while being small enough to not contaminate the turbulent flow with large coherent structures. Then, to trigger the transition, a random perturbation on the density ($\rho' = \rho\big(1 + 0.002 * A(i,j,k)\big)$, with A a Gaussian filtered white noise, and ρ the density field, see [82]) is injected into the boundary layer, the support of the noise injection is 130 cells in the streamwise direction and 15 cells in the wall normal direction, starting from the inlet and the wall respectively. As such, it is confined to the viscous region near the wall. This noise seeds all the instabilities of the streaks, which then quickly break down to turbulence. Thanks to that method, turbulent state is reached after around $\sim 37\delta_1$, with δ_1 taken at the beginning of the convergent section, of flat plate with minimal noise emission. The flow then has $\sim 90\delta_1$ to regularize before the start of the actual geometry under study. This process is qualitatively illustrated in Figure 16, showing an instantaneous snapshot of the creation of the streaks and their breakdown. Figure 17 presents the boundary layer profile and the streamwise velocity fluctuations statistics in the central $z = 0$ plane just before the first ramp ($x = 0$), both display a behavior typical of low-Reynolds number wall turbulence, with no spurious fluctuations coming from the turbulence generation method. The DNS profile is very close to the SA-QCR profile, except in the buffer region, where the model is known to behave poorly.

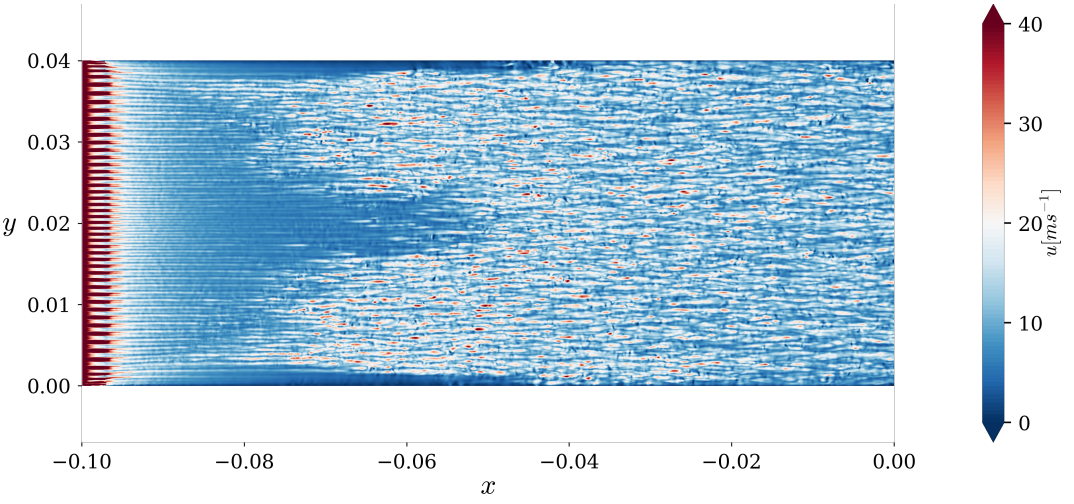


Figure 16. Streamwise velocity in the first cell in the wall normal direction, from an instantaneous snapshot of the DNS, illustrating the turbulence generation method, with first the inviscid patches to create the streaks, and then the quick breakdown to turbulence.

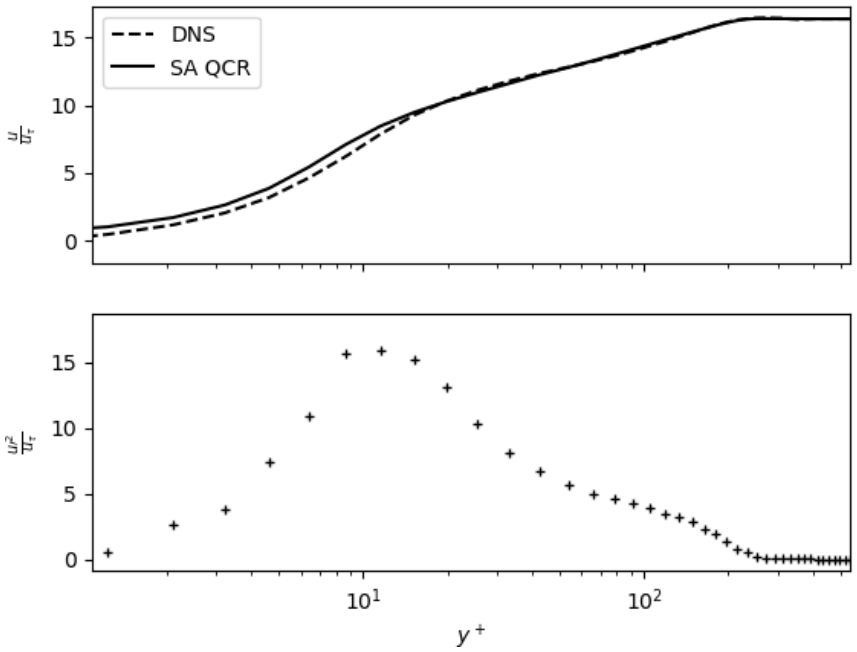


Figure 17. Time-averaged streamwise velocity profile (top) and streamwise velocity fluctuation root-mean-square value (bottom) showing the good behavior of the turbulence generation. The profile is extracted from the time-averaged flow just before the compression ramp ($x = 0$) in the central $z = 0$ plane. DNS data are sampled one point over four.

Appendix D. Impact of the turbulence model on the results

While outside the scope of the study, different turbulence models have been benchmarked before choosing the SA-QCR one. The benchmarked RANS turbulence models are the following: the Spalart–Allmaras [60] model with the compressibility correction of [61] (labeled SA) and the SST model of Menter [83] (labeled as k-ω SST). For both models, the impact of the Quadratic Constitutive Relation (labeled QCR) [62] is also tested. This non-linear addition is known to drastically improve the accuracy of linear eddy viscosity turbulence models for corner flows [84].

Figure 18 presents the Mach number in the central z-plane for each cases (the DNS correspond to time-averaged data). On the impact of the turbulence modeling strategy, one can notice that the non-QCR simulations predict a completely different separated region topology compared to the QCR ones and the DNS. The first separation point is located much more upstream, and it is caused by a weak, poorly defined shock-wave, which may be a sign that the separation comes from strong 3D effects at the corners of the geometry. The k-ω simulation quickly displays a larger separated region than the SA one. The use of QCR strongly improves the behavior of the turbulence models and leads to results in the central plane that are much closer to the DNS reference, with a separation point just upstream of $x = 0.15$ and a separation shock that is a well-defined oblique shock. The main differences between all the QCR simulations and the DNS is that the shock-train is a bit more confined to the top part of the diffuser and the low-speed region in the lower downstream part of the diffuser is larger. However, the results of the QCR RANS simulations are surprisingly close to the reference, demonstrating once again the accuracy improvement provided by the QCR on RANS simulations in the presence of corner flow.

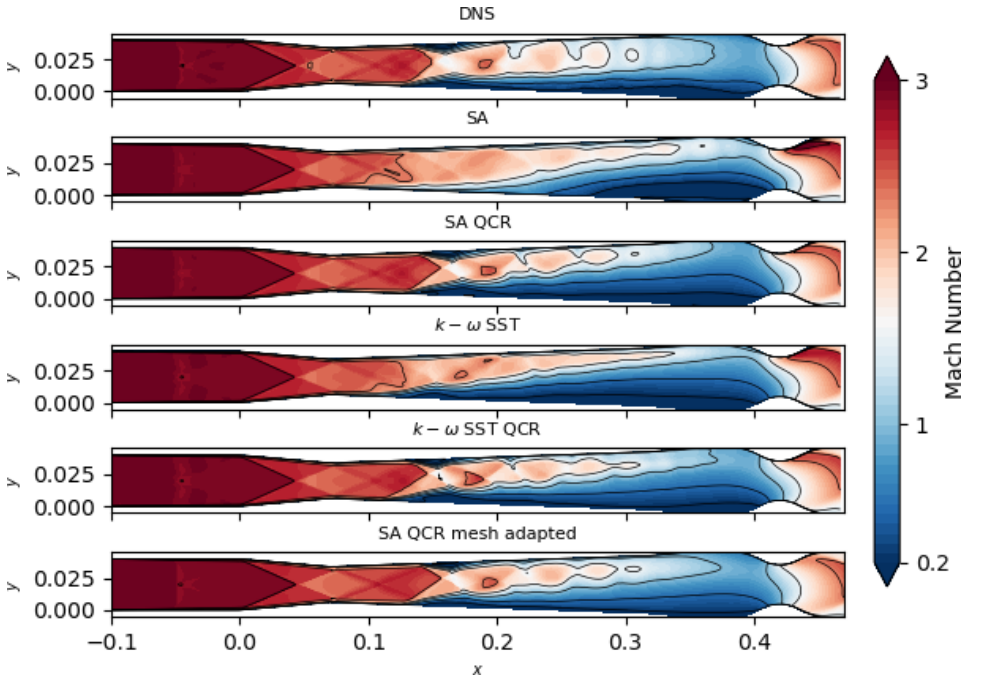


Figure 18. Mach number distribution in the central z-plane. The DNS results correspond to time-averaged flow.

References

[1] R. Mittal and G. Iaccarino, "Immersed boundary methods", *Annu. Rev. Fluid Mech.* **37** (2005), no. 1, pp. 239–261.

[2] R. Verzicco, "Immersed boundary methods: historical perspective and future outlook", *Ann. Rev. Fluid Mech.* **55** (2023), no. 1, pp. 129–155.

[3] M. Terracol and L. Manueco, "A dynamic linearized wall model for turbulent flow simulation: towards grid convergence in wall-modeled simulations", *J. Comput. Phys.* **521** (2025), article no. 113555.

[4] T.-H. Shih, L. A. Povinelli and N.-S. Liu, "Application of generalized wall function forcomplex turbulent flows", *J. Turbul.* **4** (2003), no. 1, article no. 015.

[5] A. Loseille, A. Dervieux and F. Alauzet, "Fully anisotropic goal-oriented mesh adaptation for 3D steady Euler equations", *J. Comput. Phys.* **229** (2010), no. 8, pp. 2866–2897.

[6] F. Alauzet and A. Loseille, "High-order sonic boom modeling based on adaptive methods", *J. Comput. Phys.* **229** (2010), no. 3, pp. 561–593.

[7] A. Loseille, A. Dervieux, P. J. Frey and F. Alauzet, "Achievement of global second order mesh convergence for discontinuous flows with adapted unstructured meshes", in *18th AIAA Computational Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, 2007.

[8] D. A. Venditti and D. L. Darmofal, "Grid Adaptation for Functional Outputs: Application to Two-Dimensional Inviscid Flows", *J. Comput. Phys.* **176** (2002), no. 1, pp. 40–69.

[9] F. Alauzet and L. Frazza, "Feature-based and goal-oriented anisotropic mesh adaptation for RANS applications in aeronautic and aerospace", *J. Comput. Phys.* **439** (2021), article no. 110340.

[10] C. Tarsia Morisco, E. Guilbert, M. Maunoury and F. Alauzet, "Chapter Two - Practical use of metric-based anisotropic mesh adaptation for industrial applications", in *Error Control, Adaptive Discretizations, and Applications, Part 3* (F. Chouly, S. P. A. Bordas, R. Becker and P. Omnes, eds.), Advances in Applied Mechanics, vol. 60, Elsevier, 2025, pp. 57–111.

[11] M. J. Castro-Díaz, F. Hecht, B. Mohammadi and O. Pironneau, "Anisotropic unstructured mesh adaption for flow simulations", *Int. J. Numer. Methods Fluids* **25** (1997), no. 4, pp. 475–491.

[12] A. Dervieux, D. Leservoisier, P.-L. George and Y. Coudière, "About theoretical and practical impact of mesh adaptation on approximation of functions and PDE solutions", *Int. J. Numer. Methods Fluids* **43** (2003), no. 5, pp. 507–516.

[13] P. J. Frey and F. Alauzet, "Anisotropic mesh adaptation for CFD computations", *Comput. Methods Appl. Mech. Eng.* **194** (2005), no. 48, pp. 5068–5082.

[14] C. Gruau and T. Coupez, "3D tetrahedral, unstructured and anisotropic mesh generation with adaptation to natural and multidomain metric", *Comput. Methods Appl. Mech. Eng.* **194** (2005), no. 48, pp. 4951–4976.

[15] X. Li, M. S. Shephard and M. W. Beall, "Accounting for curved domains in mesh adaptation", *Int. J. Numer. Methods Eng.* **58** (2003), no. 2, pp. 247–276.

[16] K. J. Fidkowski and D. L. Darmofal, "Output-Based Error Estimation and Mesh Adaptation in Computational Fluid Dynamics: Overview and Recent Results", *AIAA J.* **49** (2011), no. 4, pp. 673–694.

[17] G. Rogé and L. Martin, "Goal-oriented anisotropic grid adaptation", *Comptes Rendus. Mathématique* **346** (2008), no. 19–20, pp. 1109–1112.

[18] F. Hecht, "Error indicator and mesh adaption, in FreeFem++", 2009. Online at https://www.ljll.fr/hecht/ftp/FH-Mamerm09.pdf. Notes for the 3rd International Conference on Approximation Methods and numerical Modeling in Environment and Natural Resources (MAMERN '09).

[19] A. Loseille, *Génération et Adaptation de Maillages pour le Calcul Scientifique*, Université Paris-Saclay (France), 2020.

[20] C. Caillaud, A. Scholten, J. Kuehl, et al., "Separation and transition on a cone-cylinder-flare: computational investigations", *AIAA J.* **63** (2025), no. 7, pp. 2615–2634.

[21] E. K. Benitez, J. S. Jewell and S. P. Schneider, "Separation bubble variation due to small angles of attack for an axisymmetric model at Mach 6", in *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics, 2021.

[22] L. Sombaert, M. Lugrin, R. Bur, et al., "Ground tests on bolt at Mach 7: Cross-facility comparison and stability analysis", *AIAA J.* **63** (2025), no. 6, pp. 2135–2150.

[23] B. C. Chynoweth, A. Lay and J. S. Jewell, "Boundary-Layer Transition on BOLT in High Reynolds Number Mach-6 Quiet Flow", *J. Spacecr. Rock.* (2025).

[24] B. M. Wheaton, D. C. Berridge, T. D. Wolf, R. T. Stevens and B. E. McGrath, "Boundary layer transition (BOLT) flight experiment overview", in *2018 Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, 2018.

[25] J. M. Wirth, R. D. Bowersox, A. T. Dufrene and T. P. Wadhams, "Measurements of natural turbulence during the BOLT II flight experiment", *J. Spacecr. Rock.* **61** (2024), no. 5, pp. 1281–1292.

[26] C. Vu, A. Knutson, P. K. Subbareddy and G. V. Candler, "Postflight Receptivity and Transition Analysis of BOLT-II Side A Descent Trajectory", in *AIAA SCITECH 2024 Forum*, American Institute of Aeronautics and Astronautics, 2024.

[27] R. Fiévet, H. Koo, V. Raman and A. Auslender, "Numerical simulation of shock trains in a 3D channel", in *54th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2016.

[28] J. C. Dutton and B. F. Carroll, *A numerical and experimental investigation of multiple shock wave/turbulent boundary layer interactions in a rectangular duct*, technical report, University of Illinois at Urbana-Champaign, no. UILU-ENG-88-4001, 1988.

[29] L. Cambier, S. Heib and S. Plot, "The Onera elsA CFD software: Input from research and feedback from industry", *Mechanics and Industry* **14** (2013), no. 3, pp. 159–174.

[30] D. Gueyffier, S. Plot and M. Soismier, "SoNICS: a new generation CFD software for satisfying industrial users needs", 2022. Online at https://hal.science/hal-04045165/document. Conference paper: OTAN/STO/Workshop AVT-366.

[31] A. Balan, M. A. Park, S. Wood and W. K. Anderson, "Verification of anisotropic mesh adaptation for complex aerospace applications", in *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, 2020.

[32] U.S. Department of Energy (DOE), *DOE Exascale Requirements Review — Basic Energy Sciences (BES)*, technical report, U.S. Department of Energy, Office of Science, no. DOE/SC-0187, 2017. Online at https://blogs.anl.gov/exascaleage/wp-content/uploads/sites/67/2017/05/DOE-ExascaleReport_BES_R48.pdf.

[33] P. L. Roe, "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes", *J. Comput. Phys.* **43** (1981), pp. 357–372.

[34] S.-s. Chen, C. Yan, K. Zhong, H.-c. Xue and E.-l. Li, "A novel flux splitting scheme with robustness and low dissipation for hypersonic heating prediction", *Int. J. Heat Mass Transfer* **127** (2018), pp. 126–137.

[35] F. Renac, "Entropy stable, robust and high-order DGSEM for the compressible multicomponent Euler equations", *J. Comput. Phys.* **445** (2021), article no. 110584.

[36] V. Venkatakrishnan, "Convergence to Steady State Solutions of the Euler Equations on Unstructured Grids with Limiters", *J. Comput. Phys.* **118** (1995), no. 1, pp. 120–130.

[37] G. D. van Albada, B. van Leer and J. W. W. Roberts, "A comparative study of computational methods in cosmic gas dynamics", *Astron. Astrophys.* **108** (1982), no. 1, pp. 76–84.

[38] S. Piperno and S. Depeyre, "Criteria for the design of limiters yielding efficient high resolution TVD schemes", *Comput. Fluids* **27** (1998), no. 2, pp. 183–197.

[39] G. K. W. Kenway, C. A. Mader, P. He and J. R. R. A. Martins, "Effective adjoint approaches for computational fluid dynamics", *Prog. Aerosp. Sci.* **110** (2019), article no. 100542.
[40] L. N. Trefethen, A. E. Trefethen, S. C. Reddy and T. A. Driscoll, "Hydrodynamic Stability Without Eigenvalues", *Science* **261** (1993), no. 5121, pp. 578–584.
[41] F. Alizard, S. Cherubini and J.-C. Robinet, "Sensitivity and optimal forcing response in separated boundary layer flows", *Phys. Fluids* **21** (2009), no. 6, article no. 064108.
[42] B. J. McKeon and A. S. Sharma, "A critical-layer framework for turbulent pipe flow", *J. Fluid Mech.* **658** (2010), pp. 336–382.
[43] D. Sipp and O. Marquet, "Characterization of noise amplifiers with global singular modes: the case of the leading-edge flat-plate boundary layer", *Theor. Comput. Fluid Dyn.* **27** (2013), no. 5, pp. 617–635.
[44] G. Rigas, D. Sipp and T. Colonius, "Nonlinear input/output analysis: application to boundary layer transition", *J. Fluid Mech.* **911** (2021), article no. A15.
[45] B. Bugeat, J. C. Chassaing, J.-C. Robinet and P. Sagaut, "3D global optimal forcing and response of the supersonic boundary layer", *J. Comput. Phys.* **398** (2019), article no. 108888.
[46] J. R. R. A. Martins and A. B. Lambe, "Multidisciplinary Design Optimization: A Survey of Architectures", *AIAA J.* **51** (2013), no. 9, pp. 2049–2075.
[47] G. K. W. Kenway and J. R. R. A. Martins, "Multipoint High-Fidelity Aerostructural Optimization of a Transport Aircraft Configuration", *J. Aircraft* **51** (2014), no. 1, pp. 144–160.
[48] E. J. Parish and K. Duraisamy, "A paradigm for data-driven predictive modeling using field inversion and machine learning", *J. Comput. Phys.* **305** (2016), pp. 758–774.
[49] K. Duraisamy, G. Iaccarino and H. Xiao, "Turbulence Modeling in the Age of Data", *Ann. Rev. Fluid Mech.* **51** (2019), pp. 357–377.
[50] Z. Li, H. Zhang, S. C. C. Bailey, J. B. Hoagg and A. Martin, "A data-driven adaptive Reynolds-averaged Navier–Stokes k-ω model for turbulent flow", *J. Comput. Phys.* **345** (2017), pp. 111–131.
[51] L. Franceschini, D. Sipp and O. Marquet, "Mean-flow data assimilation based on minimal correction of turbulence models: Application to turbulent high Reynolds number backward-facing step", *Phys. Rev. Fluids* **5** (2020), article no. 094603 (24 pages).
[52] S. Bai, J. Z. Kolter and V. Koltun, "Deep Equilibrium Models", in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds.), Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019.
[53] M. Lienhardt and B. Michel, "SoNICS: design and workflow of a new CFD software", *Comptes Rendus. Mécanique* **353** (2025), pp. 1261–1287.
[54] C. Debiez and A. Dervieux, "Mixed-element-volume MUSCL methods with weak viscosity for steady and unsteady flow calculations", *Comput. Fluids* **29** (2000), no. 1, pp. 89–118.
[55] B. van Leer, "Towards the Ultimate Conservative Difference Scheme", *J. Comput. Phys.* **135** (1997), no. 2, pp. 229–248.
[56] F. Alauzet and L. Frazza, "Feature-based and goal-oriented anisotropic mesh adaptation for RANS applications in aeronautics and aerospace", *J. Comput. Phys.* **439** (2021), article no. 110340.
[57] L. Sciacovelli, D. Passiatore, P. Cinnella and G. Pascazio, "Assessment of a high-order shock-capturing central-difference scheme for hypersonic turbulent flow simulations", *Comput. Fluids* **230** (2021), article no. 105134.
[58] Y. Liu, B. Diskin, H. Nishikawa, W. K. Anderson, G. C. Nastac, E. J. Nielsen, A. Walden and L. Wang, "Edge-Based Viscous Method for Mixed-Element Node-Centered Finite-Volume Solvers", *AIAA J.* **62** (2024), no. 1, pp. 209–230.
[59] H. Song, K. V. Matsuno, J. R. West, A. Subramaniam, A. S. Ghate and S. K. Lele, "Scalable parallel linear solver for compact banded systems on heterogeneous architectures", *J. Comput. Phys.* **468** (2022), article no. 111443.
[60] P. R. Spalart and S. R. Allmaras, "One-equation turbulence model for aerodynamic flows", *Rech. Aerosp.* **439** (1994), no. 1, pp. 5–21.
[61] S. Deck, P. Duveau, P. d'Espiney and P. Guillen, "Development and application of Spalart–Allmaras one equation turbulence model to three-dimensional supersonic complex configurations", *Aerosp. Sci. Technol.* **6** (2002), no. 3, pp. 171–183.
[62] P. R. Spalart, "Strategies for turbulence modelling and simulations", *Int. J. Heat Fluid Flow* **21** (2000), no. 3, pp. 252–263.
[63] R. Haimes and M. Drela, "On the construction of aircraft conceptual geometry for high-fidelity analysis and design", in *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, American Institute of Aeronautics and Astronautics, 2012.
[64] J. Dannenhoffer, "An overview of the engineering sketch pad", in *AIAA SCITECH 2024 Forum*, American Institute of Aeronautics and Astronautics, 2024.
[65] H. Si, "TetGen, a Delaunay-based quality tetrahedral mesh generator", *ACM Trans. Math. Softw.* **41** (2015), no. 2, article no. 11 (36 pages).

[66] J. Schöberl, "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules", *Comput. Vis. Sci.* **1** (1997), no. 1, pp. 41–52.

[67] C. Rumsey, B. Wedan, T. Hauser and M. Poinot, "Recent updates to the CFD general notation system (CGNS)", in *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*, American Institute of Aeronautics and Astronautics, 2012.

[68] ONERA, *onera/Maia: Distributed algorithms and manipulations over CGNS meshes*, 1.8. Online at https://github.com/onera/Maia (accessed on December 10, 2025).

[69] ONERA, "Maia CGNS trees", in *Maia documentation*, 2021. Online at https://onera.github.io/Maia/1.8/introduction/introduction.html#maia-cgns-trees (accessed on December 10, 2025).

[70] ONERA, *onera/paradigm: Library for managing meshes in a massively parallel distributed environment*, 2024. Online at https://github.com/onera/paradigm/ (accessed on December 11, 2025).

[71] B. Andrieu, B. Maugars and E. Quémerais, "Dynamic load-balanced point location algorithm for data mapping", 2025. To appear in *Comptes Rendus. Mécanique*.

[72] A. Loseille and F. Alauzet, "Continuous mesh framework part II: validations and applications", *SIAM J. Numer. Anal.* **49** (2011), no. 1, pp. 61–86.

[73] A. Poulain, C. Content, D. Sipp, G. Rigas and E. Garnier, "BROADCAST: A high-order compressible CFD toolbox for stability and sensitivity using Algorithmic Differentiation", *Comput. Phys. Commun.* **283** (2023), article no. 108557.

[74] R. A. Baurle, J. A. White, M. D O'Connell, T. G. Drozda and A. T. Norris, *VULCAN-CFD User Manual: Ver. 7.2. 0*, technical memorandum, NASA, no. NASA/TM-20220008781, 2022.

[75] G. Puigt, V. Golliet and F. Alauzet, "Metric-based mesh adaptation for hypersonic flows: capturing wall heat flux with anisotropic triangles and tetrahedrons", 2025. Conference paper: The 4th International Conference on High-Speed Vehicle Science & Technology.

[76] G. V. Candler, H. B. Johnson, I. Nompelis, V. M. Gidzak, P. K. Subbareddy and M. Barnhardt, "Development of the US3D code for advanced compressible and reacting flow simulations", in *53rd AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, 2015.

[77] ONERA, *Fast: a compressible flow solver python package*, 4.0, 2024. Online at https://onera.github.io/Fast/ (accessed on December 11, 2025).

[78] I. Mary and P. Sagaut, "Large Eddy Simulation of Flow Around an Airfoil Near Stall", *AIAA J.* **40** (2002), no. 6, pp. 1139–1145.

[79] M.-S. Liou, "A sequel to AUSM, Part II: AUSM+-up for all speeds", *J. Comput. Phys.* **214** (2006), no. 1, pp. 137–170.

[80] A. Poulain, C. Content, A. Schioppa, P. Nibourel, G. Rigas and D. Sipp, "Adjoint-based optimisation of time- and span-periodic flow fields with Space-Time Spectral Method: Application to non-linear instabilities in compressible boundary layer flows", *Comput. Fluids* **282** (2024), article no. 106386 (17 pages).

[81] C. Hader and H. F. Fasel, "Towards simulating natural transition in hypersonic boundary layers via random inflow disturbances", *J. Fluid Mech.* **847** (2018), article no. R3.

[82] M. Lugrin, S. Beneddine, C. Leclercq, E. Garnier and R. Bur, "Transition scenario in hypersonic axisymmetrical compression ramp flow", *J. Fluid Mech.* **907** (2021), article no. A6.

[83] F. R. Menter, *Improved two-equation k-omega turbulence models for aerodynamic flows*, technical memorandum, NASA, no. NASA-TM-103975, 1992.

[84] J. Dandois, "Improvement of corner flow prediction using the quadratic constitutive relation", *AIAA J.* **52** (2014), no. 12, pp. 2795–2806.

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus. Mécanique
2026, Vol. 354, p. 53-70
https://doi.org/10.5802/crmeca.335

Research article

Dynamic load-balanced point location algorithm for data mapping

Bastien Andrieu [a,*], **Bruno Maugars** [b] **and Eric Quémerais** [a]

[a] ONERA/DMPE, Université Paris-Saclay, 92320 Châtillon, France

[b] ONERA/DAAA, Université Paris-Saclay, 92320 Châtillon, France

E-mails: bastien.andrieu@onera.fr, bruno.maugars@onera.fr, eric.quemerais@onera.fr

Abstract. Data mapping between geometric domains with non-matching discretizations is an essential step in multi-component numerical simulation workflows. This paper presents a novel point location algorithm, designed for transferring data from unstructured meshes to point clouds, in a massively parallel distributed environment. Special emphasis is placed on load balancing, which is paramount for making the most of computing resources and achieve optimal performance. In general, the geometric entities of interest are unevenly distributed in the input frame provided by the calling codes. The algorithm therefore aims to rapidly prune the search space using a series of parallel preconditioning techniques, while redistributing data equitably across all processes at each step. Exact point-in-cell location is then computed in an embarrassingly parallel, well-balanced frame. All data movements performed throughout the point location algorithm are transparent to the calling codes, as the resulting geometric and parallel mappings are returned in the same frame as the input data. These mappings enable data transfer via spatial interpolation and optimized process-to-process communications. A weak scaling study is carried out in three scenarios representative of the variety of real-life applications. Comparison with a state-of-the-art algorithm shows that the new algorithm performs better overall, with speed-ups of up to a factor of 10 on 4,800 CPU cores.

Keywords. High Performance Computing (HPC), Message Passing Interface (MPI), dynamic load balancing, computational geometry.

Note. Article submitted by invitation.

Manuscript received 15 January 2025, revised 5 October 2025, accepted 20 October 2025, online since 3 February 2026.

1. Introduction

In many scientific and engineering applications, numerical simulations require transferring data between arbitrarily discretized domains. Such applications include code coupling for multi-physics simulations [1,2], solution transfer following adaptive remeshing [3], and Lagrangian particle tracking [4]. Depending on the different numerical methods, these spatial discretizations usually consist of meshes or point clouds.

Data transfer from a *source* (donor) domain to a *target* (receiver) domain breaks down to two main steps. First, a mapping between the source and target degrees-of-freedom (DoFs) must be computed. When treating the target DoFs as points, the task consists in identifying which cell of the source mesh contains each of these points. Such points typically correspond to mesh

* Corresponding author

nodes or quadrature points in the Finite Element and Discontinuous Galerkin methods, or cell centers in the Finite Volume method. Second, the source-to-target mapping is used to interpolate and transfer data. The location step is the most computationally intense, but only needs to be performed once at initialization if both source and target geometries remain static during the simulation. However, data transfer between domains with dynamic discretizations requires repeated location computations, making the performance of this operation crucial. Ideally, the time needed for data mapping should be within the same order of magnitude as the time required for a single iteration of a computational code. Naively testing all possible pairs of cells and points results in quadratic complexity, which is prohibitively expensive in practical applications. It is therefore essential to devise efficient preconditioning techniques that eliminate all the irrelevant pairs.

The growing demand for high-fidelity simulations requires an ever-increasing number of degrees-of-freedom, and simulations exceeding a billion DoFs are becoming more prevalent [5]. Such simulations require so much memory and computing power that they can only be achieved on massively parallel distributed-memory systems, using parallel programming models such as the Message Passing Interface (MPI) [6]. In this context, the domains between which data is to be transferred are decomposed into partitions distributed across multiple processes.

This parallel context poses an additional challenge. The domain decompositions are tailored for the specific needs of each computational code and are therefore not optimal for the location problem. Since the source and target partitions are unlikely to align, extra communication is necessary to compute point location and exchange interpolated data. To prevent communication latency from becoming a bottleneck, the preconditioning stage must also minimize unnecessary communications. Besides, the geometric entities of interest may be poorly distributed in the decompositions provided as input to the location algorithm. For instance, when transferring data through the common surface between two adjacent volume domains, only a fraction of processes hold a portion of this surface in their partition. If no action is taken to redistribute the workload, this imbalance can severely degrade performance.

The distribution and amount of input data can vary significantly across different data mapping applications. Ensuring good performance regardless of this variability is challenging and developing a single algorithm that runs efficiently in all situations remains an open problem.

The rest of this paper is structured as follows. Section 2 gives a brief overview of existing work on data transfer for massively parallel simulations. A novel parallel point location algorithm tailored for data transfer between distributed meshes and point clouds is then described in Section 3. Finally, a performance study of this algorithm in several test cases representative of the variety of real applications is reported in Section 4, along with comparisons with another high-performance algorithm.

2. Related work

The *precise Code Interaction Coupling Environment* (preCICE) library [7] is a state-of-the-art open-source coupling library which features multiple data transfer methods. The underlying location algorithm, recently improved by Totounferoush et al. [8], can be broken down into two major steps. The first step consists in comparing the axis-aligned bounding boxes (AABBs) of the mesh partitions owned by each process, in order to establish the pairs of processes from the two coupled codes that will need to communicate. The AABBs are initially collected by a master process for each coupled code. They are then transmitted to the other master process and subsequently broadcast to all the remaining processes involved in the computation. In the second step, source mesh partitions are exchanged to the corresponding target processes using a process-to-process communication pattern. Each process then locates its own target points

within the received source mesh partitions. This is performed efficiently using an R-Tree data structure. While showing great improvement over their previous algorithm, some limitations still remain. Most of the total execution time is spent exchanging and comparing the AABBs in the first step and communicating source mesh partitions to the target processes. The load imbalance issue is also not addressed. In fact, the performance studies shown assume meshes uniformly distributed on all processes, which is unlikely in real-life surface coupling applications.

The *Finite Volume Mesh* (FVM) library [9] features location and exchange capabilities, leaving the interpolation step to the calling codes for better genericity. The location algorithm follows essentially the same two steps as preCICE, albeit with noticeable strategic differences. First, the partition AABBs are exchanged via collective communications, thus avoiding the bottleneck inherent to the master-slave approach implemented in preCICE. Second, the location computation is performed on the source side, meaning each process sends its target points to its corresponding source processes. This approach reduces the amount of communication, since smaller messages are needed to exchange points instead of mesh partitions. In order to keep the memory requirements low, blocking send/receive communications are used and the received point clouds are located one at a time. The location step is then accelerated by storing the received points in a local octree. Fournier [10] notes that this strategy can lead to excessive serialization in worst-case scenarios. To solve this problem, a technique for ordering communicating processes by recursive subdivision is implemented in the *Parallel Location and Exchange* (PLE) library, which is derived from FVM. Fournier also points out that the first coarse-grained filter based on a single AABB per process can lead to the communication of a large number of potentially irrelevant points, due to numerous false positive detections. Assuming a uniformly distributed point cloud, the number of points received by each process is correlated to the volume of its AABB. Yet this volume highly depends on the shape of the partitions and can vary considerably from one process to another, resulting in significant load imbalance. Possible optimizations are proposed to address this issue, including the use of a distributed box-tree data structure, enabling finer filtering. Once calculated by the calling code, the interpolated data is finally exchanged following the same communication pattern as in the first stage of the location algorithm. All pairs of processes which partition AABBs intersect thus communicate, even if no points from one process have effectively been located in the mesh partition held by the other. Load imbalance and serialization can therefore compromise this step as well.

The *Data Transfer Kit* (DTK) library [11] addresses the load imbalance issue by creating a secondary *rendezvous* decomposition [12], well balanced for the location problem. Points and cells outside the domain bounded by the intersection of the global source and target AABBs are first discarded. Recursive coordinate bisection is then performed on the combined source and target geometries. The MPI communicator is also recursively split along the way, leaving in the end one partition per process, each containing geometrically close source cells and target points. The splitting procedure aims to balance the combined number of source cells and target points. Some rendezvous partitions can thus end up containing virtually only source cells and no target points, or vice versa. This worst-case scenario can occur if the level of refinement of both source and target discretizations differ significantly. Geometric location in the rendezvous decomposition is accelerated using geometric binning or a local *k*d-tree. The authors also propose to perform the interpolation step in a well-balanced fashion in the same rendezvous decomposition, at the expense of additional communications required to transfer the source field data from the initial decomposition to the rendezvous decomposition.

The algorithm presented in the next section aims to remedy the shortcomings mentioned above, by developing a more refined preconditioning strategy and ensuring good load balance at each critical step.

3. Point location algorithm

3.1. *Key concepts*

Before presenting this algorithm, the following paragraphs define the terms used in this paper and outline the key concepts at the core of our point location algorithm.

3.1.1. *Point location problem*

We focus on data transfer between unstructured meshes and point clouds, with interpolation schemes using stencils restricted to the source cell containing each target point. Other types of interpolation with wider stencils (e.g. k nearest neighbors, radial basis functions, etc.) would require a different preconditioning strategy, which will be studied in future work.

Given a set S of *source* cells and a set T of *target* points, point location consists in finding for each point in T the *host* cell from S in which it lies, if any.

The two sets S and T are initially distributed on P processes that form an arbitrary MPI communicator. All subsequent communications will occur within this communicator, which is provided as an input.

3.1.2. *Frames*

Dynamic load balancing is essential to achieve high performance on distributed-memory systems, and involves redistributing data across processes throughout the algorithm. In this paper, the different data distributions are called *frames*. If E designates a set of entities, the part of E held by process p in frame $\mathscr{F}$ will be denoted by $E_p^{\mathscr{F}}$. The size of this part is denoted by $|E_p^{\mathscr{F}}|$. The *input* frame will be denoted by $\mathscr{I}$. In order to remain as generic as possible, no particular assumption is made on this frame. Some parts may contain more entities than others, some may even be empty. Some entities may also be held by multiple processes, as in the case of ghost cells or mesh vertices located on boundaries between adjacent partitions.

3.1.3. *Global identifiers*

To keep track of an entity (a source cell, its bounding box or a target point) moving across different frames, our algorithm relies on *global* identifiers (IDs). Contrary to the *local* IDs used within each partition, global IDs are *frame-independent* and *unique*, meaning that if two entities on different processes share the same global ID they are in fact two instances of the same entity. This is essential for achieving reproducible results that do not depend on how data is distributed in the input frame. Such global numbering can either be provided as an input, or generated from any ordered data set.

3.1.4. *Dynamic load balancing framework*

Dynamic load balancing is traditionally achieved using graph-based partitioning methods since they generally yield simply connected partitions with minimal edge cut. While these properties are essential for most computational codes, they are of little interest for solving the point location problem in parallel. When partition connectedness is not an issue, parallel sorting algorithms provide a more cost-effective solution for redistributing the workload. Global IDs are also used extensively for this purpose in our algorithm. Entities are re-partitioned by assigning each process an equal-sized block of entities with contiguous global IDs. This is achieved by sorting entities globally in ascending order of IDs using parallel bucket sort. If entities have associated weights, a parallel bucket sampling algorithm[1] is used in order to devise blocks of equal weight prior to sorting.

[1] An implementation on the GPU of this algorithm is presented in [13].

3.1.5. *Subsets*

Some input data might not be relevant to the location problem, e.g. cells that are guaranteed to contain no points. Our algorithm aims to isolate the *subsets* of interest by quickly pruning these irrelevant entities. Building new global numberings restricted to such subsets provides better conditioning for the sorting algorithms used for dynamic load balancing. Communication graphs are associated to each subset in order to enable data movement between the different frames. To make this possible, an explicit link between the subset and parent global numberings are maintained throughout the different steps of our algorithm. Table 1 illustrates this concept.

Table 1. Illustration of a subset (second row) and its parent set (first row) distributed on three processes (each process is represented by a color). The subset is described in two different frames: $\mathscr{F}_1$ which is balanced with respect to the parent set, and $\mathscr{F}_2$ which is balanced with respect to the subset. Frame $\mathscr{F}_2$ is obtained by redistributing the subset entities in ascending order of parent global IDs. This order is preserved in the global numbering proper to the subset.

	Frame $\mathscr{F}_1$												Frame $\mathscr{F}_2$					
Global ID in parent set	3	7	1	9	2	6	10	12	5	8	4	11	4	6	7	8	10	11
Global ID in subset	–	3	–	–	–	2	5	–	–	4	1	6	1	2	3	4	5	6

3.2. *Algorithm outline*

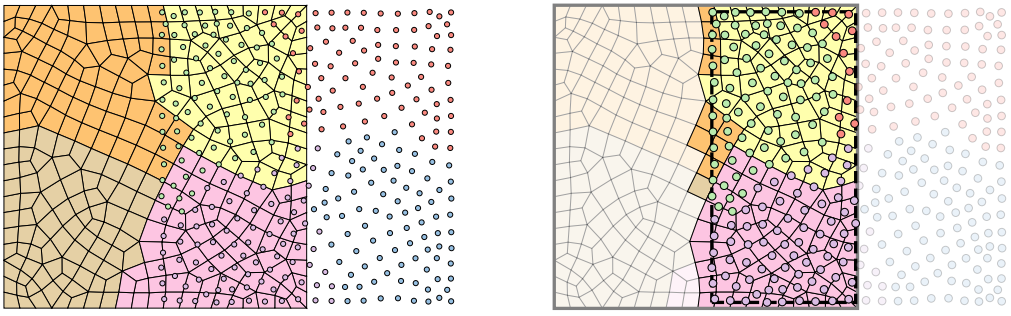
The point location algorithm presented in this paper is structured in the following five main steps. A first coarse-grained filter is followed by a second, finer preconditioning step based on fast point-in-box tests. This search for candidate cell-point pairs is accelerated using a distributed tree structure, which allows to redistribute evenly the location workload. The third step consists in computing exact point-in-cell location for all candidate pairs. Potential conflicts are then resolved in a fourth step, in order to retain at most one host cell per point. Last, the source-to-target process-to-process communication channels are established for subsequent exchanges of interpolated data.

These five steps are detailed in the next sections and illustrated with a basic, two-dimensional example.

3.3. *Coarse filtering*

The first step in our point location algorithm consists in a coarse filtering similar to the one proposed by Plimpton et al. [12]. The aim is to quickly eliminate cells and points that are clearly irrelevant to the location problem. Each process computes the AABB of the source cells in its partition. A collective reduction operation is then performed to obtain the global AABB of S, denoted by $\underline{S}$ (shown in solid gray line in Figure 1(b)). Let T' denote the subset of target points located inside $\underline{S}$. These are the only points that can possibly be located inside a source cell. The global AABB $\underline{T'}$ of T' is then computed in a similar way (shown in dashed black line in Figure 1(b)). Let S' denote the subset of cells whose AABB intersects $\underline{T'}$. These are the only cells that can possibly contain target points. Cells (resp. points) not in S' (resp. T') will no longer be considered in the remainder of the algorithm. This step proves useful when the source and target geometries overlap only partially (such as in the example of Figure 1), yet induces very little overhead if they overlap completely, as will be highlighted in Section 4.

At this stage, each process holds a (possibly unequal) part of each subset S' and T', respectively denoted by $S'^{\mathscr{I}}_p$ and $T'^{\mathscr{I}}_p$. In the example shown in Figure 1, S' is distributed over almost only two processes.



(a) Whole partitions.

(b) Subsets S' and T' in the same frame. The solid gray and dashed black rectangles depict the global AABBs of S and T', respectively.

Figure 1. Source mesh and target point cloud partitions in the input frame $\mathscr{I}$ (each color represents a process).

3.4. *Search for candidate pairs*

The search for candidate cell-point pairs relies on inexpensive point-in-box tests. However, the naive strategy that consists in checking all possible pairs local to each process becomes prohibitively expensive for large numbers of cells and points. Besides, at this stage, subsets S' and T' are arbitrarily distributed in the input frame, and their respective parts generally do not align.

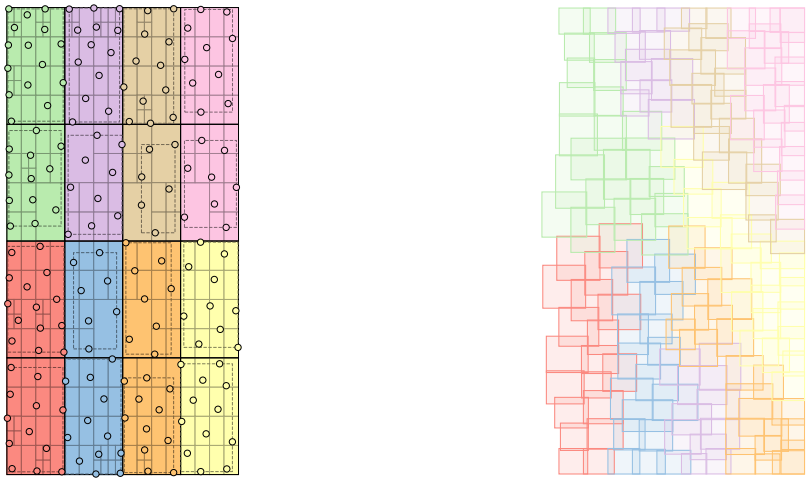
Therefore, each process must identify to which other processes each of its points or boxes should be sent. This filtering is also based on AABB comparisons. To address the issues raised in Section 2, our algorithm relies on multiple boxes per process. This more refined filter helps reducing the amount of data exchanged in this step.

Sundar et al. [14] present a method for constructing distributed linear octrees that naturally yields such boxes. This octree structure can also be leveraged to speed up the second stage of the candidate search, local to each process.

In principle, the distributed box-tree proposed by Fournier [10] is quite similar to this structure. Both are constructed following a bottom-up approach, ensuring a good load balance using re-partitioning based on the Morton space-filling curve [15]. However, whereas the box-tree requires fine-tuning of four parameters with complex combined effects, the octree proposed by Sundar is governed by just two parameters: the maximum depth of the tree and the maximum number of points contained in each of its leaf nodes.

3.4.1. *Octree construction*

The algorithm for constructing the distributed octree [14] consists in the following main steps. Target points are first sorted globally in ascending Morton order and redistributed evenly to obtain a new frame $\mathscr{O}$. In this frame, all partitions $\{T'^{\mathscr{O}}_p\}$ contain an equal amount of points. This first step is further detailed in Section 3.4.3. The points are then sorted locally and converted into octree leaves at maximum depth. Then, a minimal linear octree is constructed in parallel by filling in the gaps between successive leaves by empty octants. Finally, the octree is coarsened so as to partition the point cloud into as few coarse *blocks* as possible while maintaining good load balance. From each of these coarse blocks stems a finer sub-tree local only to the process owning that block. This algorithm ensures that the number of blocks per process is between 1



(a) Distributed quadtree and the associated target partitions in frame $\mathscr{O}$. Coarse blocks (solid black rectangles) and their effective AABBs (dashed rectangles) are shown, along with the sub-trees local to each process.

(b) AABBs of the source cells in the Morton SFC-based balanced frame $\mathscr{B}$.

Figure 2. SFC-based frames for target and source entities.

and 8. These sub-trees share a common ancestry, thus forming a complete, distributed octree. Such an octree (in fact a two-dimensional quadtree) is represented in Figure 2(a).

3.4.2. *Octree query*

Once octree construction is complete, our algorithm proceeds as follows. The coarse blocks are gathered on all processes so that each process can determine to which other processes it should send the source AABBs it owns. A given source AABB is sent to a process if it intersects at least one of the coarse blocks owned by this process. In order to further reduce the amount of false positive detections, each block is reduced to the AABB of the points it contains. Such AABBs are shown as dashed rectangles in Figure 2(a).

As the number of processes increases, the memory and CPU cost of storing and querying these blocks can become problematic. CPU overhead can be amortized by storing the blocks in a tree structure, thus enabling queries in $O\big(|S'_p|\log(P)\big)$ time. This operation is trivial, since the blocks are natively obtained from a coarse octree. The shared coarse tree associated to the example in Figure 2(a) is represented in Figure 3. Besides, a single instance of this coarse tree can be stored on each compute node, taking advantage of shared memory. On typical supercomputers with tens of cores per compute node, the memory footprint of the coarse tree can thus be reduced by at least an order of magnitude.

Querying the coarse tree thus provides a connection between each source AABB and its processes of interest. This connection is used to exchange the boxes through collective MPI communications. As a result, each process receives a partition $S'^{\mathscr{O}}_p$ of source AABBs in the octree frame $\mathscr{O}$. The part of the octree local to this process is then inspected in order to find candidate points for each of these boxes.

Despite the acceleration provided by the tree structure, querying the coarse blocks can become a critical step if the partitions $S'^{\mathscr{I}}_p$ are unbalanced. This is typically the case in the example

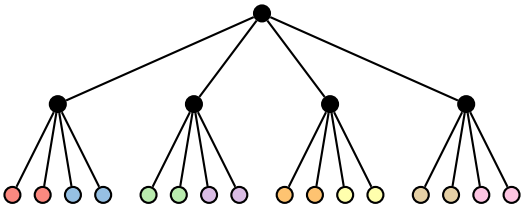


Figure 3. Schematic view of the shared coarse tree. Each leaf corresponds to a coarse block, colored by its owner process.

shown in Figure 1(b). Moreover, an arbitrary distribution of boxes can result in a highly dense communication graph, significantly increasing the latency of collective communications.

To address these issues, the source AABBs are redistributed before querying the shared coarse tree. A good heuristic is to apply the same sorting and redistribution as performed on the points in the first step of octree construction. The new, balanced frame thus obtained is denoted by $\mathscr{B}$. As shown in Figure 2(b), the source partitions $\{S_p'^{\mathscr{B}}\}$ are mostly aligned with the target partitions $\{T_p'^{\mathscr{O}}\}$. The next paragraph briefly describes how this re-partitioning is achieved.

3.4.3. *SFC-based re-partitioning*

Given an arbitrarily distributed set of geometric entities (i.e. points or boxes), the goal is to find a new, well-balanced distribution that maximizes data locality. Space filling curves (SFC) have been used extensively for this purpose [16,17].

SFCs map continuously points in three-dimensional space to one-dimensional space. These curves have inherent multi-resolution properties due to their iterative construction, making them attractive for constructing linear tree structures. Setting a maximum resolution reduces space to a finite number of voxels. The cartesian coordinates of a point can then be encoded as a nonnegative integer, corresponding to its voxel index, dictated by the traversal order of the SFC. The Morton SFC (or "Z-order curve") [15] provides a straightforward encoding that can be very efficiently computed using bit shifts. The Morton ordering is equivalent to that obtained by traversing an octree depth first.

Subsets S' and T' are re-partitioned independently, but in the same way. The strategy described in Section 3.1.4 is followed, except that Morton codes are used instead of arbitrary global IDs. These Morton codes are computed by encoding either the target point coordinates or the coordinates of the source AABB centers. The subset global IDs thus obtained correspond to the order in the globally sorted array of Morton codes.

The re-partitioning induces a new balanced frame for each subset S' and T', to which data must be communicated from the input frame. The coordinates of the target points are communicated, along with only the AABBs of the source cells. The search for candidate pairs indeed requires no additional information such as mesh connectivity or vertex coordinates. To obtain the final source-to-target mapping in the input frame, as discussed in Section 3.7, it is essential to maintain an explicit link between subset entities and their corresponding parent sets. The global IDs of entities in the parent sets S and T are therefore communicated to the new frames as well.

Since S' and T' are re-partitioned independently from each other, the exchange of point data can be overlapped by computations for finding the balanced distribution of source cells, using non-blocking collective communications. Moreover, the exchange of cell data can be overlapped by the construction of the distributed octree.

3.5. *Exact point-in-cell location*

Once the octree query is complete, a connection is obtained between the source cells and their candidate points. This connection is represented in the form of a graph that is distributed in the octree frame $\mathscr{O}$.

The next step consists in computing the exact location of these candidate points for each cell. Weights for subsequent interpolation of source data at target points must also be calculated. Applications based on the Finite Element method usually rely on shape function interpolation. To this end, the isoparametric coordinates of each candidate point are determined using Newton–Raphson iteration. When dealing with ill-conditioned shape functions (e.g. highly curved high-order elements), this iteration may fail to converge. In this case a second, more robust technique is used, which consists in recursively subdividing the cell after a first decomposition into simplices. When dealing with meshes composed of arbitrary polygonal or polyhedral cells, a first inclusion test determines whether each candidate point lies inside or outside the cells. In the first case, mean value coordinates [18,19] are calculated to serve as interpolation weights. In the second case, the nearest projection onto the cell's faces is computed, along with the distance from this projection.

These exact location computations could be carried out in the frame $\mathscr{O}$. However, this frame is not guaranteed to be well balanced. Some processes may indeed hold more cells than others, or cells with more candidate points. Load balance in this stage is critical since computing exact location is expensive. In addition, these computations require a full description of the source cells, i.e. mesh connectivity along with vertex coordinates, whereas only AABBs are available at this point in frame $\mathscr{O}$. This additional information is only available in the input frame $\mathscr{I}$.

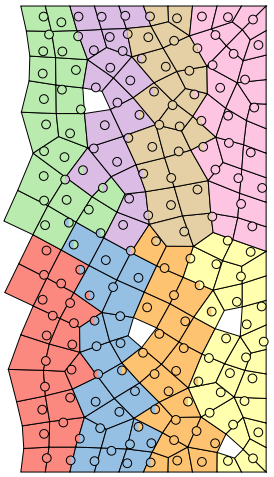
The choice is thus made to create a new, better balanced rendezvous frame in which exact location will be computed. In order to keep data movements to a minimum, the rendezvous frame is created such that each cell is owned by a single process. Points may in turn be duplicated on multiple processes. The number of candidate points in the AABB of each cell gives a simple yet effective estimation of the workload associated to that cell. This way, cells with no candidate points can simply be discarded. The different cell types could also be taken in consideration to further refine this estimate.

This rendezvous frame is obtained using the approach described in Section 3.1.4 by sorting cells in ascending order of the subset global IDs derived from the SFC-based re-partitioning. The data locality provided by this global numbering reduces duplication of mesh entities (e.g. faces, vertices) since adjacent cells are likely to be owned by the same process in the rendezvous frame, as indicated in Figure 4(a).

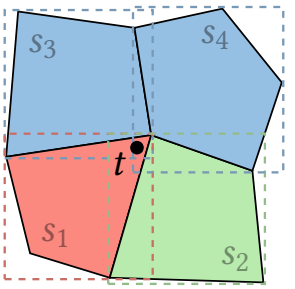
3.6. *Conflicts resolution*

Each target point may be contained in the AABB of multiple source cells. Besides, these cells may reside on different processes in the rendezvous frame, and therefore so do the target points. In the end, each point must be mapped to at most one host cell. Possible conflicts are resolved by filtering the location data computed in the previous step in order for each point to keep only the best match among its associated candidate cells. This filtering is performed in two stages. First, the candidates local to each process are sorted in ascending order of signed distance.[2] The nearest candidate in that process is then kept, while the others are discarded. The second stage is similar, except that this time the sorting is carried out on the best candidates identified by all the processes in the previous stage. If a point is located exactly between two cells, the conflict is resolved by selecting the cell with lower global ID. This way, the location algorithm is guaranteed

[2] A negative distance indicates that the point is contained inside the cell.



(a) Partitions of candidate cells and points in the rendezvous frame $\mathscr{R}$. Points duplicated on multiple processes and are shown in multiple colors.



(b) Location conflict for a point with multiple candidate cells (colored by owning process in frame $\mathscr{R}$).

Figure 4. Exact point-in-cell location steps.

to yield reproducible results, that do not depend on the number of processes. In order to balance the workload, this second stage is performed in a new frame $\mathscr{C}$.

The approach described in Section 3.1.4 is used once again to devise such a balanced frame. Distance values of the cells that pass the first filter are exchanged from the rendezvous frame $\mathscr{R}$ to the conflict-resolution frame $\mathscr{C}$. The result of the second filter is then sent back to frame $\mathscr{R}$ so that each cell can discard candidate points for which it is not the best match.

In the situation depicted in Figure 4(b), a target point t is associated to four candidate cells, distributed on three processes in frame $\mathscr{R}$ (indicated by colors). On the blue process, the first filter retains the nearest cell, namely s_3. Distances from s_1, s_2 and s_3 are then communicated to the process holding t in frame $\mathscr{C}$. The second filter finally retains s_1.

3.7. *Return to the input frame*

At this stage, all false positive detections have been eliminated. The remaining cell-point pairs form the actual mapping used for transferring data between the source and target domains. However, this data is usually distributed in the same frame as the input source and target partitions, namely frame $\mathscr{I}$. This data could be first communicated from frame $\mathscr{I}$ to the rendezvous frame $\mathscr{R}$, in which interpolation would be performed, as suggested by Plimpton et al. [12]. Nevertheless, frame $\mathscr{R}$ may no longer be well balanced since the connection between cells and points has just been pruned. Besides, additional application-specific information may be required for spatial interpolation, and would need to be communicated to frame $\mathscr{R}$ as well. Experience shows that the interpolation step is about two orders of magnitude less expensive than the location step. Possible load imbalance due to poor distribution in the input frame therefore does not compromise performance. Sticking to frame $\mathscr{I}$ for interpolation is thus a sensible choice. The source-to-target mapping is thus transferred from the rendezvous frame to the input frame.

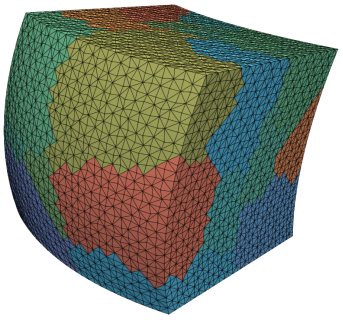
A process-to-process communication pattern is finally established between the source and target partitions in this input frame, to enable subsequent exchanges of interpolated data.

4. Performance results

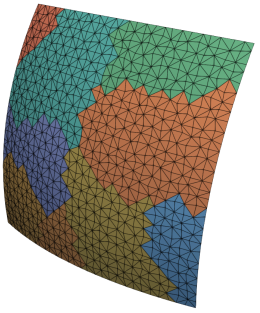
The point location algorithm presented in this paper has been implemented in the open-source CWIPI coupling library [20,21], along with high-level wrappings to MPI primitives to carry out the transfer of interpolated data using non-blocking send/receive communications, based on the previously described communication pattern. CWIPI formerly relied on the FVM library for data mapping, using the algorithm mentioned in Section 2. The tests presented in this section were carried out in version 1.1.0 of CWIPI [21] which incorporates both algorithms, thereby enabling a direct comparison.

4.1. *Test cases description*

The following setup is considered to study and compare the performance of the two point location algorithms. Starting from a cartesian grid, an unstructured source mesh is obtained by decomposing the hexahedral cells into tetrahedra. To break the alignment with the cartesian axes, the geometry is deformed and a slight random perturbation is applied to the vertex coordinates, as shown in Figure 5.



(a) Volume source mesh used in the full and partial volume overlap scenarios.



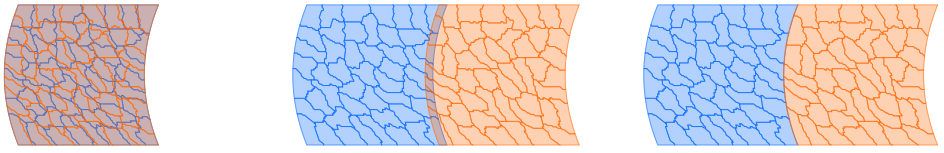
(b) Surface mesh extraction used in the surface overlap scenario.

Figure 5. Meshes used in the tests (colors indicate the different partitions).

A second mesh is generated using the same procedure, and its cell centers are employed as the target point cloud. Grids with slightly different numbers of points are used so that the two meshes do not match exactly. The source and target meshes are partitioned using the PT-Scotch library [22] on two separate MPI communicators, each with the same number of processes. The communicators are then merged by CWIPI into a single COMM_WORLD communicator. Since the partitioning method used is not deterministic, the sensitivity of both algorithms to input distributions is examined by executing each test case multiple times. A weak scaling study is carried out, with roughly $2.5 \cdot 10^5$ cells (resp. points) in each source (resp. target) partition. First, a source-to-target mapping is constructed using the point location algorithms. Then, a field evaluated on the source mesh is interpolated and transferred to the target points using either the communication scheme proposed by FVM or the one described in the end of Section 3.7. Interpolation here consists simply in assigning each target point the field value of its host cell, as the study focuses primarily on the performance of the parallel algorithm (the

use of a more refined scheme such as shape-function interpolation would have no impact on overall performance, since the interpolation weights are calculated regardless). The code for reproducing the test cases (as well as the input meshes, which are automatically generated at runtime) is available online (see Section 5 for the details).

Different scenarios representative of the wide range of real-life applications are realized by offsetting the meshes relative to each other in the x-direction, as represented in Figure 6. For each of the three configurations considered, execution times for the location and interpolation steps are reported for both algorithms. Minimum, mean and maximum times over the multiple executions of each test are reported. Finally, a detailed breakdown of the location time is presented for the new algorithm.



(a) Full volume overlap. **(b)** Partial volume overlap. **(c)** Surface overlap.

Figure 6. Schematic view of the three scenarios. The partitioned source and target domains are shown in blue and orange, respectively.

4.2. *Hardware description*

All the tests presented in this paper were carried out on the same supercomputer. Each of its compute nodes is composed of two 2.4 GHz Intel Xeon 6240R (Cascade Lake) processors, each one containing 24 cores, totalling 48 cores per node. The nodes are interconnected by an Intel OmniPath 100 GB/s low-latency network. All experiments were run using one MPI rank per core.

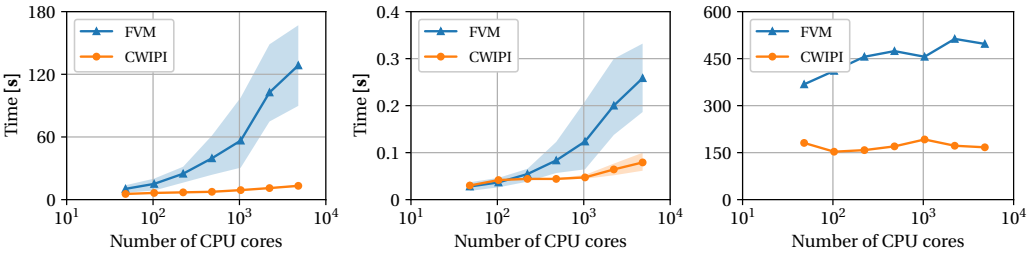
4.3. *Performance analysis*

4.3.1. *Full volume overlap*

In the first test case, the source and target geometries overlap completely. This scenario arises in situations such as solution transfer before restarting a simulation on a different mesh. In this case, the coarse filtering step described in Section 3.3 reveals useless, since the global source and target AABBs are essentially identical. However, it induces very little overhead since it accounts for less than 1 % of the total execution time. This step is therefore omitted in the breakdown given in Figure 8.

Although the source mesh is evenly distributed, the volume of the partition AABBs can vary considerably between processes. In the worst case, a volume ratio of 300 is observed between the largest and smallest AABBs. The number of points to be located by each process in the FVM algorithm is equally unbalanced, as anticipated in Section 2. Furthermore, Figure 7 shows that the execution time of the FVM algorithm varies significantly between different runs, indicating a high sensitivity to the input partitioning. In contrast, the new algorithm delivers the same execution time regardless of the partitioning, and exhibits greater parallel efficiency. This improvement is explained by the finer preconditioning (Section 3.4) and the redistribution of the location workload (Section 3.5). A maximum load imbalance of 10 % is observed in the rendezvous frame.

Figure 7(b) demonstrates the benefits of optimizing the communication graph used to exchange the interpolated data. The strategy described in Section 3.7 proves reasonable, considering the small proportion of execution time devoted to the "return to input frame" step, as shown in Figure 8. Figure 7(c) also confirms that the interpolation step is significantly cheaper than point location (by about two orders of magnitude).



(a) Point location.

(b) Interpolation and exchange.

(c) Ratio point location / interpolation and exchange.

Figure 7. Execution times for the FVM and CWIPI algorithms (full volume overlap). Average times over all runs are represented by solid lines, while shaded areas illustrate the range of elapsed times across all runs.

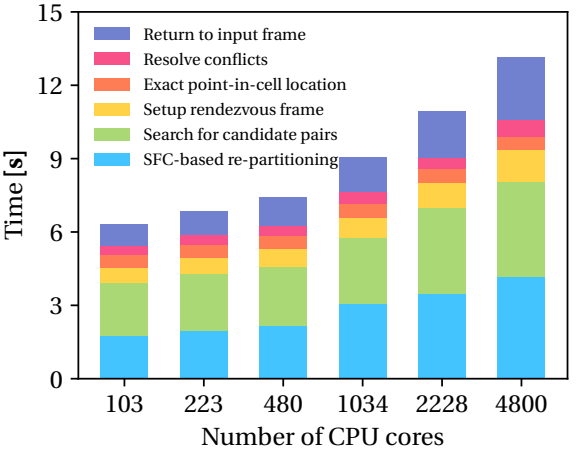


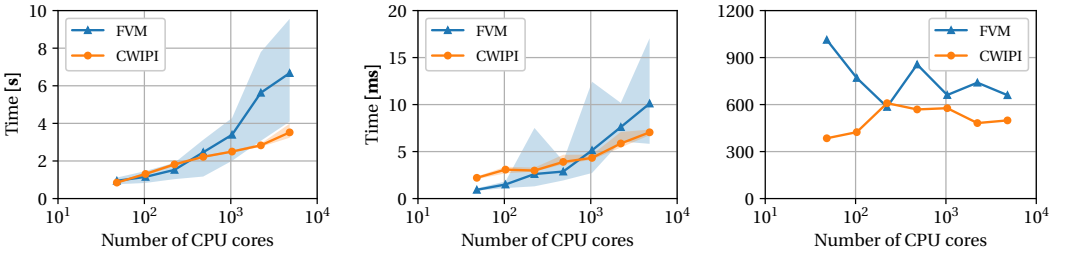
Figure 8. Breakdown of average execution time for the new point location algorithm (full volume overlap).

In applications with dynamic geometries, point location must be performed frequently. Higher frequency generally translates into greater fidelity and numerical robustness. The CPU cost of data mapping should ideally be comparable to that of one iteration of a computational code, which is on the order of 1 µs per cell for state-of-the art computational fluid dynamics solvers. In comparison, the CPU cost of the new point location algorithm ranges from 20 µs to 50 µs per target point in this test. Dynamic simulations with large meshes thus remain challenging, and optimizations are necessary to overcome the data mapping bottleneck. As indicated in Figure 8, these optimizations should focus primarily on the preconditioning stage.

4.3.2. *Partial volume overlap*

In the second test case, the target point cloud is shifted so that it overlaps only a thin layer of the source mesh (a few cells thick). This scenario mimics applications such as the multi-component simulation presented in [23], in which data mapping is used to integrate the different moving parts of a full aircraft engine into a single unsteady large-eddy simulation.

Full source and target geometries are provided to the location algorithms, letting them filter out irrelevant points and cells. Figure 9 shows that variations in the input partitioning have a strong impact on the performance of the FVM algorithm, as in the first test case. Besides, most processes are idle during the location step, since the AABB of their source mesh partition intersect no target partition AABB. This load imbalance results in suboptimal parallel efficiency.



(a) Point location. **(b)** Interpolation and exchange. **(c)** Ratio point location / interpolation and exchange.

Figure 9. Execution times for the FVM and CWIPI algorithms (partial volume overlap). Average times over all runs are represented by solid lines, while shaded areas illustrate the range of elapsed times across all runs.

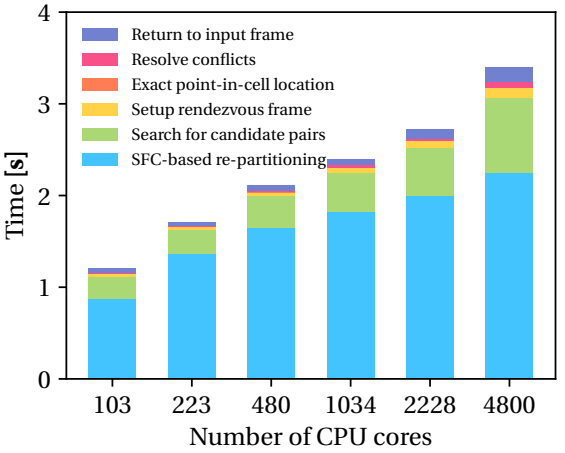


Figure 10. Breakdown of average execution time for the new point location algorithm (partial volume overlap).

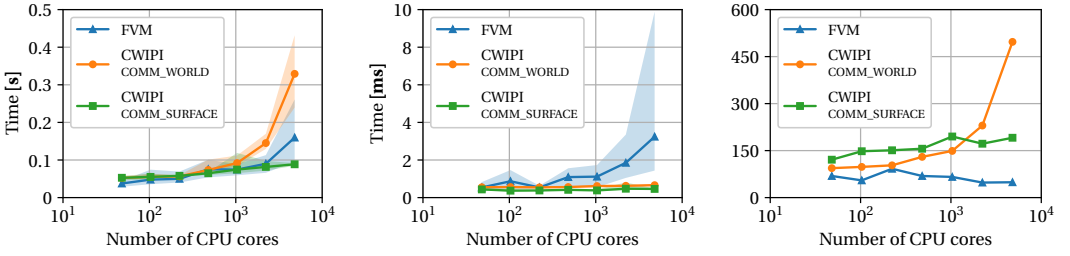
The new algorithm effectively eliminates up to 95 % of the source cells and target points in the first coarse filtering step, once again with minimal extra cost. Dynamic load balancing proves less effective than in the first test case, since the average workload per process is considerably reduced. In fact, the SFC-based re-partitioning step accounts for about 70 % of the total execution time, as reported in Figure 10. A more detailed analysis reveals that the bucket sampling algorithm is responsible for 60 % of the cost of this step. This algorithm relies on parameters

tuned using heuristics, which could be further optimized for our application. The communication graph used by the FVM algorithm to transfer the interpolated data is much sparser than in the first case. Consequently, both algorithms yield comparable execution times for this step. Still, the new point location algorithm performs better on average and remains much less sensitive to the input data distribution. In this second test, the location step takes around 10 µs per target point[3] with the new algorithm, which is still quite expensive compared to an iteration of fluid simulation.

4.3.3. *Surface overlap*

In the third test case, the target point cloud is shifted so that it overlaps only one boundary face of the source domain. This scenario is representative of surface couplings such as simulation of fluid-structure interaction [1]. In such applications, data transfer is typically used to apply specific boundary conditions. Only the surface geometric entities and their associated DoFs are therefore considered.

As in the other two test cases, volume meshes are first generated and partitioned. The surface of interest is then extracted without subsequent redistribution, as shown in Figure 5(b). Consequently, the source cells and target points are unevenly distributed in this input frame, since the surface is contained in only a fraction of the volume partitions. This fraction diminishes as the number of processes increases. Nevertheless, the deterioration in load imbalance is tempered by a decrease in the average number of cells and points per partition, which scales as $O(P^{-1/3})$. In this situation, communication latency becomes predominant. The new algorithm carries out numerous data movements and is therefore penalized.



(a) Point location. **(b)** Interpolation and exchange. **(c)** Ratio point location / interpolation and exchange.

Figure 11. Execution times for the old and new algorithms (surface overlap). Average times over all runs are represented by solid lines, while shaded areas illustrate the range of elapsed times across all runs.

A solution to improve parallel efficiency consists in using fewer processes so that the increased average workload compensates for the communication overhead. This strategy is tested by splitting the initial MPI communicator (here COMM_WORLD), to obtain the sub-communicator COMM_SURFACE restricted to processes holding a non-empty share of the surface in the input frame. Figures 11 and 12 demonstrate the effectiveness of this strategy. In multi-component simulations such as in [23], multiple simultaneous data transfers need to be carried out, in relatively small, distinct geometric regions. Assigning one sub-communicator to each data transfer task would allow for a more effective utilization of computational resources. Yet, this approach could be refined by determining an optimal communicator size based on an assessment of the total workload. Splitting the working communicator after the coarse filtering step could enable even further improvement. However, if data transfer is considered as part of a multi-component

[3] All the target points are considered, including the ones not contained in any cell.

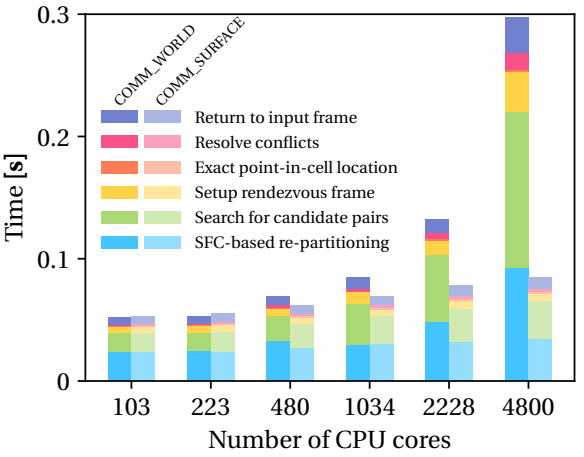


Figure 12. Breakdown of average execution time for the new point location algorithm (surface overlap), using either COMM_WORLD or COMM_SURFACE as the input MPI communicator.

simulation, performance remains acceptable (on the order of 1 µs per target point)[4] and the proposed optimizations become less significant.

5. Conclusion

In this paper a novel point location algorithm has been presented, designed for data mapping between distributed meshes and point clouds in a massively parallel environment. Comparisons with a state-of-the-art algorithm are favorable, with a speed-up of up to a factor of 10. This achievement is made possible by a more refined preconditioning strategy combined with dynamic load balancing.

Good scaling is observed for up to 1.2 billion cells and points on 4,800 CPU cores, proving that our algorithm can be integrated in a wide range of real-life, large-scale data transfer applications. In order to prepare for the exascale era, the performance analysis still needs to be carried out on larger supercomputers. For instance, the first and last step of the algorithm (see Section 3.4.3 and 3.7 respectively) rely heavily on collective (all-to-all) MPI communications which could become a bottleneck with a higher number of MPI ranks.

Work is also underway to harness the full potential of heterogeneous architectures and accelerate the preconditioning stage through CPU-GPGPU hybridization [13]. The expected speed-up should help overcome the current bottleneck experienced in dynamic simulations. Finally, the possible optimizations proposed in this paper to mitigate the communication overhead will also be explored in future work.

Acknowledgments

The manuscript was written through contributions of all authors.

[4] The whole *volume* mesh is considered.

Underlying data

The tests presented in Section 4 can be reproduced by running the Python script `python_perfo_location_octree.py` available in the `tests/` directory after cloning CWIPI's GitHub repository: https://github.com/onera/cwipi/blob/master/tests/python_perfo_location_octree.py.

Declaration of interests

The authors do not work for, advise, own shares in, or receive funds from any organization that could benefit from this article, and have declared no affiliations other than their research organizations.

References

[1] T. Fabbri, G. Balarac, V. Moureau and P. Benard, "Design of a high fidelity Fluid–Structure Interaction solver using LES on unstructured grid", *Comput. Fluids* **265** (2023), article no. 105963.

[2] G. Coria, J.-D. Parisse, J.-M. Lamet and N. Dellinger, "Modeling and simulation of chemical reactions at the surface of an ablative wall interacting with a hypersonic flow", 2022. Conference paper: 9th European Conference for Aeronautics and Aerospace Sciences (EUCASS-3AF).

[3] F. Alauzet, "A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes", *Comput. Methods Appl. Mech. Eng.* **299** (2016), pp. 116–142.

[4] A. Palha, L. Manickathan, C. S. Ferreira and G. van Bussel, "A hybrid Eulerian-Lagrangian flow solver", 2015. Online at https://arxiv.org/abs/1505.03368.

[5] A. W. Cary, J. Chawner, E. P. Duque, W. Gropp, W. L. Kleb, R. M. Kolonay, E. Nielsen and B. Smith, "CFD Vision 2030 Road Map: Progress and Perspectives", in *AIAA AVIATION 2021 FORUM*, American Institute of Aeronautics and Astronautics, 2021.

[6] The MPI Forum, "MPI: a message passing interface", in *Proceedings of the 1993 ACM/IEEE conference on Supercomputing* (B. Borchers and D. Crawford, eds.), ACM Press, 1993, pp. 878–883.

[7] G. Chourdakis, K. Davis, B. Rodenberg, et al., "preCICE v2: A sustainable and user-friendly coupling library", *Open Res. Eur.* **2** (2022), article no. 51 (47 pages).

[8] A. Totounferoush, F. Simonis, B. Uekermann and M. Schulte, "Efficient and scalable initialization of partitioned coupled simulations with preCICE", *Algorithms* **14** (2021), no. 6, article no. 166 (17 pages).

[9] Y. Fournier, J. Bonelle, C. Moulinec, Z. Shang, A. G. Sunderland and J. C. Uribe, "Optimizing Code_Saturne computations on Petascale systems", *Comput. Fluids* **45** (2011), no. 1, pp. 103–108.

[10] Y. Fournier, "Massively parallel location and exchange tools for unstructured meshes", *Int. J. Comput. Fluid Dyn.* **34** (2020), no. 7–8, pp. 549–568.

[11] S. Slattery, P. Wilson and R. Pawlowski, "The Data Transfer Kit: A geometric rendezvous-based tool for multiphysics data transfer", in *International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering (M&C 2013)*, 2013, pp. 5–9.

[12] S. J. Plimpton, B. Hendrickson and J. R. Stewart, "A parallel rendezvous algorithm for interpolation between multiple grids", *J. Parallel Distrib. Comput.* **64** (2004), no. 2, pp. 266–276.

[13] R. Cazalbou, F. Duchaine, E. Quémerais, B. Andrieu, G. Staffelbach and B. Maugars, "Hybrid Multi-GPU Distributed Octrees Construction for Massively Parallel Code Coupling Applications", in *PASC '24 — Proceedings of the Platform for Advanced Scientific Computing Conference*, ACM Press: Zurich, Switzerland, 2024.

[14] H. Sundar, R. S. Sampath and G. Biros, "Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel", *SIAM J. Sci. Comput.* **30** (2008), no. 5, pp. 2675–2708.

[15] G. M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*, IBM, 1966.

[16] S. Aluru and F. E. Sevilgen, "Parallel domain decomposition and load balancing using space-filling curves", in *Proceedings 4th International Conference on High-Performance Computing*, IEEE, 1997, pp. 230–235.

[17] R. Borrell, J. C. Cajas, D. Mira, A. Taha, S. Koric, M. Vázquez and G. Houzeaux, "Parallel mesh partitioning based on space filling curves", *Comput. Fluids* **173** (2018), pp. 264–272.

[18] K. Hormann and M. S. Floater, "Mean value coordinates for arbitrary planar polygons", *ACM Trans. Graph.* **25** (2006), no. 4, pp. 1424–1441.

[19] T. Ju, S. Schaefer and J. Warren, "Mean Value Coordinates for Closed Triangular Meshes", *ACM Trans. Graph.* **24** (2005), no. 3, pp. 561–566.

[20] E. Quémerais, "La Bibliothèque de Couplage CWIPI — Coupling With Interpolation Parallel Interface", in *ONERA, le centre français de recherche aérospatiale*, 2013. Online at https://w3.onera.fr/cwipi/bibliotheque-couplage-cwipi (accessed on November 27, 2025).

[21] ONERA, *onera/cwipi: Library for coupling parallel scientific codes via MPI communications to perform multi-physics simulations*, 1.1.0. Online at https://github.com/onera/cwipi/tree/cwipi-1.1.0 (accessed on November 27, 2025).

[22] C. Chevalier and F. Pellegrini, "PT-Scotch: A tool for efficient parallel graph ordering", *Parallel Comput.* **34** (2008), no. 6–8, pp. 318–331.

[23] C. P. Arroyo, J. Dombard, F. Duchaine, L. Gicquel, B. Martin, N. Odier and G. Staffelbach, "Towards the Large-Eddy Simulation of a Full Engine: Integration of a 360 Azimuthal Degrees Fan, Compressor and Combustion Chamber. Part I: Methodology and Initialisation", *J. Global Power Propul. Soc.* **2021** (2021), pp. 1–16.

Comptes Rendus
Mécanique

Objet de la revue

Les *Comptes Rendus Mécanique* sont une revue électronique évaluée par les pairs de niveau international, qui couvre l'ensemble des domaines des sciences mécaniques.
Ils publient des articles originaux de recherche, des articles de synthèse, des mises en perspective historiques, des textes à visée pédagogique, ou encore des actes de colloque, en anglais ou en français, sans limite de longueur et dans un format aussi souple que possible (figures, données associées, etc.).
Depuis 2020, les *Comptes Rendus Mécanique* sont publiés avec le centre Mersenne pour l'édition scientifique ouverte, selon une politique vertueuse de libre accès diamant, gratuit pour les auteurs (pas de frais de publication) comme pour les lecteurs (accès libre, immédiat et pérenne).

À propos de la revue

Toutes les informations concernant la revue, y compris le texte des articles publiés qui est en accès libre intégral, figurent sur le site https://comptes-rendus.academie-sciences.fr/mecanique/.

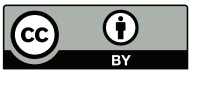
Informations à l'attention des auteurs

Pour toute question relative à la soumission des articles, les auteurs peuvent consulter le site https://comptes-rendus.academie-sciences.fr/mecanique/.

Contact

Académie des sciences
23, quai de Conti, 75006 Paris, France
Tel: (+33) (0)1 44 41 43 72
cr-mecanique@academie-sciences.fr

Pour citer ce numéro :
Nicolas Berthier, Denis Guyeffier, Éric Quémerais (ed). Multiphysics simulation environments. Comptes Rendus Mécanique, 2026. https://doi.org/10.5802/crmeca.sp.4.



Comptes Rendus
de l'Académie des sciences

Mécanique

Multiphysics simulation environments / *Les environnements de simulation multiphysiques*

Contents

Please note that the page numbers indicated are those of the electronic file of the issue. Each article also has its own pagination, linked to its publication date.

Cover illustration: Coupled multiphysics simulation of the Safran Tech Bearcat engine configuration (ONERA elsA codes for aerodynamics and CEDRE for combustion). The static temperature field is displayed on a reconstructed exploded 3D representation of the engine. / *Simulation multiphysique couplée de la configuration moteur Safran Tech Bearcat (codes ONERA elsA pour l'aérodynamique et CEDRE pour la combustion). Le champ de température statique est affiché sur une représentation 3D éclatée reconstituée du moteur.*

Authors (ONERA) : K. Anemiche, L. Castillon, G. Chaineray, L. Ruan.

1873-7234 (electronic)

Académie des sciences - Paris