L. Cambier, M. Gazaix, S. Heib, S. Plot, M. Poinot, J.-P. Veuillot
(Onera)
J.-F. Boussuge, M. Montagnac
(Cerfacs)

E-mail: laurent.cambier@onera.fr

An Overview of the Multi-Purpose *elsA* Flow Solver

Development of the *elsA* software for complex external and internal flow aerodynamics and multidisciplinary applications started in 1997 at Onera. Due to the multi-purpose nature of *elsA*, many common basic CFD features can be shared by a wide range of aerospace applications: aircraft, helicopters, turbomachinery, missiles, launchers… The *elsA* software is based on an Object-Oriented design method and on an Object-Oriented implementation based on three programming languages: C++, Fortran and Python. The *elsA* strategy for interoperability is based on a component approach that relies on standard interfaces for the CFD simulation components. This paper presents an overview of the capabilities of the *elsA* software in terms of modeling, mesh topology, numerics and boundary conditions, whereas a more detailed description of these capabilities is given in companion papers of this issue of the electronic journal. The importance of High Performance Computing activities is outlined in the present paper.

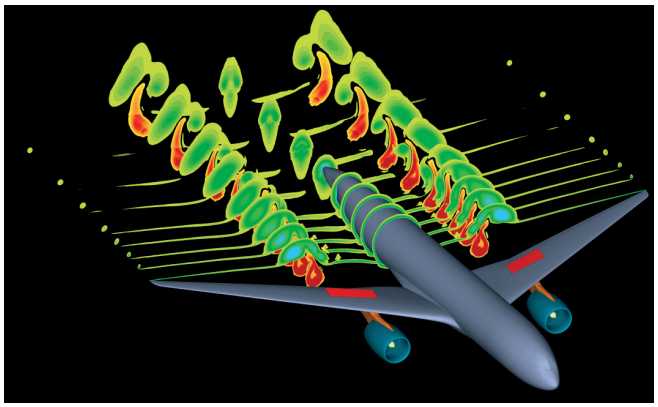
Introduction

Unlike other industries, such as the automobile industry, the simulation software used for aerodynamic analysis and design in the aeronautic industry is not usually provided by commercial software vendors, but is generally developed either by Research Establishments or sometimes by the aeronautic industry itself. The overview of software tools used for flow simulation in the European aeronautic industry, presented in [26], shows that in Europe they are mostly managed by Research Establishments. One major reason for this is that the high levels of accuracy and reliability required today for improving aeronautic design are obtained through long term expertise and innovative research in various areas: physical modeling, numerical methods, software efficiency on rapidly evolving hardware and validation by comparison with detailed experimental data.

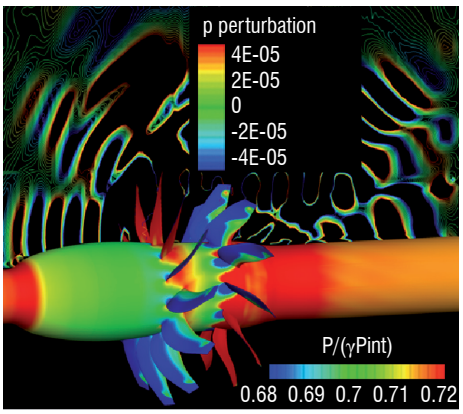
The *elsA* software (http://elsa.onera.fr) for complex external and internal flow aerodynamics and multidisciplinary applications has been developed at Onera since 1997 [4], [6]. The main objective is to offer the French and European aerospace community a tool that capitalizes on the innovative results of Computational Fluid Dynamics (CFD) research over time and is able to deal with miscellaneous industrial applications. The range of aerospace applications dealt with using *elsA* (aircraft, helicopters, tilt-rotors, turbomachinery, counter-rotating open rotors, missiles, launchers, etc.) is very wide, as shown in figure 1 presenting a few examples of *elsA* results. A large variety of the advanced aerodynamic applications handled by *elsA* is presented in [25]. Note that it is quite uncommon for a CFD tool to deal both with external flows around airframes and with internal flows in turbomachinery; since it allows common basic CFD features to be shared, we clearly consider that it is an advantage. The research, development and validation activities are carried out using a project approach in cooperation with the aircraft industry and external laboratories or universities (see Box 1). This project approach has been the result, at Onera as elsewhere, of the change in CFD software development from a one-code, one-developer paradigm at the beginning of the eighties to a team-based approach necessary to cope with the complexity of today's CFD. The effort carried out in France and coordinated by Onera with *elsA* was also initiated approximately at the same time as projects in other countries, such as the WIND-US flow solver of the NPARC Alliance [3] or the FAAST program at NASA Langley RC [19] in the United States or the TAU flow solver in Germany [14].

This paper gives a general presentation of the *elsA* solver, and mainly focuses on software topics such as Object-Oriented design and implementation, software interoperability or High Performance Computing. Only a general overview of the capabilities of the *elsA* software in terms of modeling, mesh topology, numerics, aeroelasticity and optimum design is presented here, whereas a more detailed description of these capabilities and results, showing evidence of their functionality and correctness, is given in companion papers of

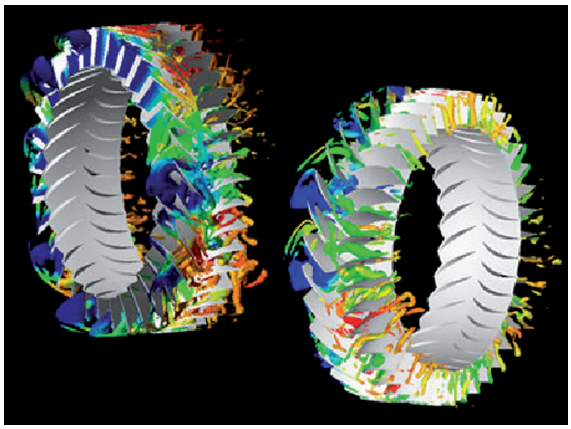
this issue of the electronic journal ([1] for transition and turbulence modeling, [8] for space discretization methods on various mesh topologies, [21] for time integration methods, [10] for aeroelasticity, [22] for optimum design). Also, the reader should refer to another companion paper [25] for a detailed presentation of the application results of *elsA*, which only appear in this paper as illustrations.



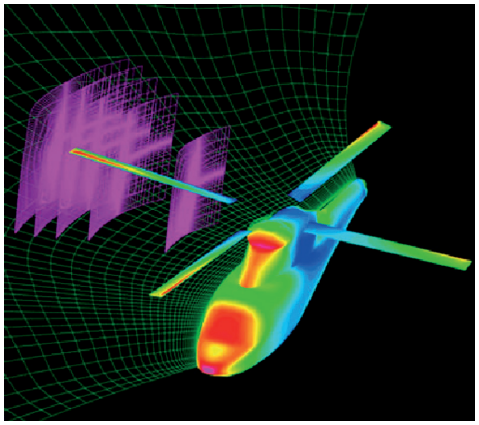
a) Transport aircraft configuration: simulation of the flow around an aircraft with deployed spoilers



c) CROR configuration: simulation of the flow around a CROR for an aero-acoustics study



b) Turbomachinery configuration: simulation of the rotating stall in an axial compressor stage



d) Helicopter configuration: simulation of the interaction between the main rotor and the fuselage

Figure 1 – Examples of applications carried out with *elsA*

Box 1 - Internal and external developers and users

Many partners, not only inside Onera but also in external laboratories and universities and in the aerospace industry, contribute to the development of new capabilities and to the validation of *elsA*. Inside Onera, the CFD and Aeroacoustics Department of Onera coordinates the *elsA* software project and contributes to the development in terms of software architecture, numerical methods or CPU efficiency. Several other Departments take part in development and validation activities related to *elsA* software: namely the Applied Aerodynamics Department for thorough validation and some specific applied aerodynamics developments, the Aerodynamics and Energetics Modeling Department for transition and turbulence modeling and fundamental validation, and the Aeroelasticity and Structural Dynamics Department for fluid/structure capability development and validation.

Since 2001, there has been a partnership with the Cerfacs research organization for *elsA* development, and Cerfacs has taken part in many developments, dealing, in particular, with mesh strategies, numerical methods and CPU efficiency, since that time. Other labs also take part in the development and validation of *elsA*, such as the Fluid Mechanics and Acoustics Lab of "École Centrale de Lyon" for development of complex turbomachinery boundary conditions, the applied research center Cenaero (Belgium) for turbomachinery flow simulation and the DynFluid lab of "Arts et Métiers ParisTech" for high accuracy numerical schemes. The use of *elsA* in French engineering schools or universities is also developing for academic teaching purposes.

elsA is today intensively used as a reliable design tool in the French and European aeronautic industry. In turbomachinery industry, *elsA* is used in the design teams of Safran group (Snecma and Turbomeca in France, Techspace Aero in Belgium). For transport aircraft configurations, *elsA* is one of the two CFD programs used at Airbus for performance prediction and for design (the other one is the TAU software from DLR, see [5]). Among other industry partners, we should mention Eurocopter for helicopter applications and MBDA for missile configurations.

Why a multi-purpose tool for solving the Navier-Stokes equations?

CFD methods and software have improved tremendously over the last forty years. Whereas in the 1970s, CFD software for design was mostly based on formulations assuming the flow to be inviscid, today CFD codes solving the Navier-Stokes equations (see Box 2) have become standard tools in the aeronautic industry [27]. The improvements concern various areas: mesh topology capabilities, physical modeling, numerical algorithms. However, in each of these areas, there is no universal method answering all of the problems. The best choice of methods depends on the type of application and on the levels of accuracy, robustness and efficiency that are required. In the case of mesh topology capabilities, the relative advantages and disadvantages of Cartesian structured grids, curvilinear structured grids and unstructured grids are well-known today and in recent years it has become clearer that an association of various types of grids in a single simulation is very powerful.

It is also well-known that transition and turbulence modeling which is required for the simulation of turbulent flows has to be adapted to the type of application and even to local flow phenomena. For example, a classical eddy viscosity model should give good results at low cost for the flow simulation around an airfoil at low angle of attack, whereas Large Eddy Simulation would be required for the simulation of the flow for near stall conditions. The selection of the best numerical algorithm strongly depends on the flow regime (subsonic, transonic or supersonic flow) and on the compromise which is required between accuracy, efficiency and robustness.

One solution could be to build dedicated software focusing on a narrow application domain. But this solution leads to a proliferation of specific software tools, which is very difficult to maintain, document, optimize and port to different computers. In fact, real-world applications today require a large range of capabilities. And if the best choice of methods depends on the type of application, it is also true that one specific method may be useful for various types of applications (see below in the section “Mesh topology capabilities” one example of this with the Chimera method). Also, the scientific community is looking for larger and more complex simulations, and it has become increasingly necessary to involve and combine several models and/or several meshing strategies in the same flow simulation.

So, CFD software designers are faced with the challenge of meeting a very wide range of requirements, while keeping software complexity and development cost under control. Thus, a very broad range of CFD capabilities has to be grouped together in an interoperable and evolving software package. To cope with these broad requirements, the designers of the *elsA* software chose to rely on an Object-Oriented design method as will be described below and *elsA* was one of the first Object-Oriented major scientific packages written in C++ [13].

This choice was quite successful since there has been an intensive development of *elsA* throughout the years. Today, *elsA* is being developed towards a component architecture (see the section dealing with interoperability) to cope with ever increasing requirements: smart integration in the simulation environments of the aeronautic industry, runtime control of the simulation, coupling with external software for multi-disciplinary applications, etc. Coupling independent components through a common high-level infrastructure provides a natural way to reduce the complexity.

Object-Oriented design and implementation

The *elsA* software is based on an Object-Oriented (OO) design method. The central concept of OO design is the class: a class encapsulates data and methods. The class interface is a way of communicating with class users. The most important difference between procedural and OO programming is the switch from function to class as the fundamental abstraction. OO programming is interesting because it makes it easier to think about programs as collections of abstractions and to hide the details of how these abstractions work to users who do not care about these details. OO programming can be used to partition problems into well-separated parts, none of which needs to know more about the others than absolutely necessary. This ability to break a large, developing program down into parts that can be pursued independently, thus enhancing collaboration among multiple contributors, explains why integration of new developments is easier in an OO software.

Box 2 - Mathematical model of *elsA*: the Navier-Stokes equations

Navier-Stokes equations are the main mathematical model of aerodynamics in the continuous regime, for which the characteristic length scales are large compared with the mean free path of the molecules. These equations result from the application of the principles of mechanics and thermodynamics, and, in integral form, express an equilibrium between a volume term expressing the time variation of mass, momentum (mass times velocity) and total energy (sum of internal energy and kinetic energy) contained in a volume Ω, and a surface flux term corresponding to the exchanges between the fluid inside Ω and the fluid outside Ω.

The Navier-Stokes equations are completed, on the one hand, by behavior laws representing the irreversibility effects associated with viscosity and thermal conductivity, and, on the other hand, by state laws describing the thermodynamic properties of the fluid.

As a result of the low viscosity of air, flows in aeronautics applications are turbulent. Turbulence is included in the Navier-Stokes equations which represent all the turbulence scales. However, Direct Numerical Simulation which relies on solving the instantaneous Navier-Stokes equations is still very far from being applicable to real world applications. Therefore, a statistical approach has to be added to the Navier-Stokes equation, which leads to Reynolds Averaged Navier-Stokes or to Large Eddy Simulation (see [1] for further details).

One important objective of *elsA* is to reap the benefits of the OO approach without impairing numerical efficiency. Since nearly all CPU time of CFD calculations is spent in computing loops acting over quantities such as cells, nodes or interfaces, the OO design of *elsA* does not deal with objects such as individual cells or fluxes, since loops operating on them would suffer a high penalty. Conversely, outside of loops, it has been shown with *elsA* design that OO concepts may be introduced without any significant CPU penalty.

The last version of *elsA* delivered includes about 600 classes grouped in 26 modules specialized for a given CFD task. The OO model based on CFD experience has been defined using the classical UML ("Unified Modeling Language") method. Ideally, developers should be able to work inside a module, without having to know the implementation details of the other modules. Achieving a good breakdown is very important for achieving ease co-operative of development and maintenance. A UML modeling tool was used to define the initial design of *elsA*, but without relying on the automatic code generation capability. The use of such a tool appeared to be too difficult for managing the cooperative development of *elsA* and was given up.

Each module in *elsA* is identified by a key of 3 to 5 letters. Inside each module, each class name is then prefixed by the key of the module to which it belongs. As an example, the `TurKL` class associated with the (k, l) turbulence model belongs to the `Tur` module which deals with turbulence modeling and transition prediction.

Some of turbulence models implemented in *elsA* are built on the Boussinesq hypothesis. Their common feature is the use of the eddy viscosity which can be calculated either by an algebraic turbulence model, a subgrid-scale model or using transport equations. Each turbulence model is implemented through a specific class, which is derived from the abstract base class `TurBase`. All the classes deriving from the abstract class `TurBase` share its interface which declares the method `compMut()`. When manipulated in terms of the interface defined by `TurBase`, the concrete classes do not have to be known by the client classes. Client classes are only aware of the abstract class. In *elsA*, the client classes of the `Tur` class hierarchy are the diffusive fluxes (see module `Fxd` below) that manipulate a pointer to an instance of a class derived from `TurBase` by means of:

```
tur -> compMut()
```

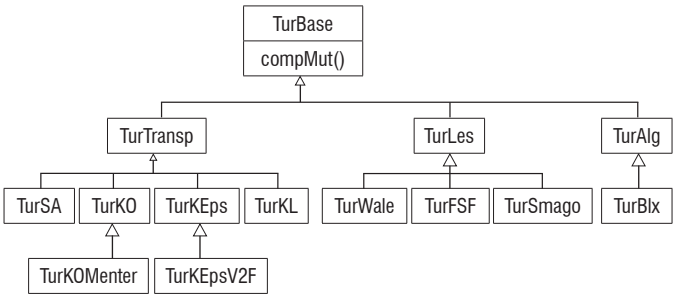


Figure 2 – UML model of the `Tur` module

The computation of the eddy viscosity depends on each particular turbulence model and cannot be performed in the `TurBase` abstract class. Polymorphism allows the correct version of `compMut()` to be called dynamically, without any explicit coding by the programmer. As a consequence, adding a new turbulence model will not modify the code of the client class. Figure 2 presents a simplified and reduced view of the UML class diagram of `Tur`.

elsA design also follows the objective of organizing the modules into layers in such a way that each layer should mainly affect the layers above (see Figure 3). That means that classes in a layer are only allowed to use services of classes of lower layers (or of same layer); the goal of this organization is to achieve mono-directional relationships. The advantage is then that maintenance becomes easier since one layer's interface only affects the next layers. We will now present the main modules of *elsA*.

The lowest layer contains all of the low level modules, such as the `Fld` module (corresponding to the data storage classes which encapsulate the dynamic memory needed to store any computational data), the `Pcm` module (dealing with the implementation on parallel computer architectures, it encapsulates the message passing interface) or the `Sio` module dealing with `IO` (Input/Output).

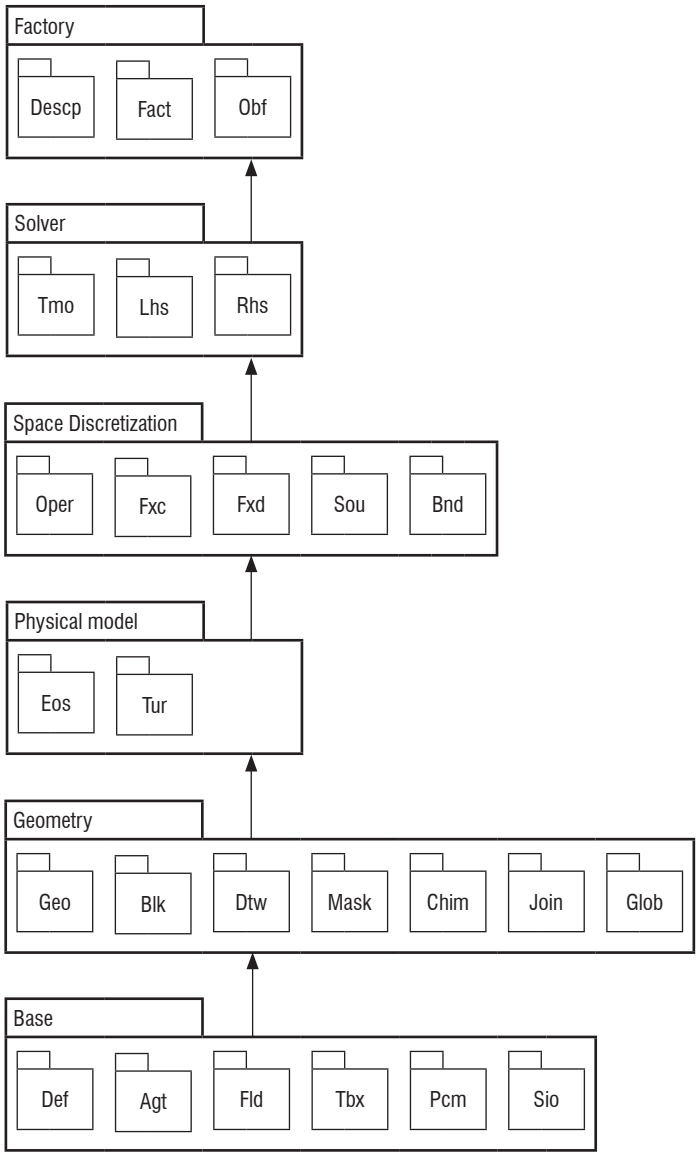


Figure 3 – *elsA* design organization

Then, the geometry layer contains all of the modules which describe geometrical and topological elements:

- `Blk`: defines the "block" notion. A block corresponds to a region of the discretized physical space defined by a mesh. Blocks are specialized to take into account grid motion, Arbitrary Lagrangian-Eulerian (ALE) technique and Hierarchical Mesh Refinement (HMR) features;

Box 3 - Joint development project

The *elsA* project is a joint development project including organizations and individuals located at several geographic places. This cooperative development involving developers at different sites is greatly facilitated by the use of a simple and robust version control system: the Concurrent Version System (CVS), and soon the Subversion system (SVN). The version control system maintains a central "repository" which stores files and gathers them into coherent and named sets of revisions. The developers work in private workspaces and use the repository as a common basis for source exchange. Only one person, called the "integrator", is allowed to "commit" the set of changes done by a developer back into the repository. Each new production of the software approximately includes 5 to 10 developments.

A delivered release of the software is a particular production, corresponding to a higher level in documentation and validation quality than a current production. Average time between two successive delivered releases of *elsA* is 12 to 18 months, and generally about 5 intermediate productions are made.

Communication is obviously one of the key elements in a joint development project. A Web site (http://elsa.onera.fr) facilitates information transmission. This site, including restricted areas for external users and developers, gives, in particular, access to the documentation (User Reference Manual, User Starting Guide, Developer's Guide, Validation report…), the validation scripts (around 170 test cases at the present time in the validation database) and the problem tracking database. A specific email address is available for software support.

- `Geo`: defines the abstraction of the computational grid and provides with all geometrical ingredients used by the finite volume formulation (metrics: volume of cells, surface of cell interfaces; topological relations between geometrical entities: cells, interfaces, nodes);
- `Dtw`: contains all of the distance and boundary-layer integral thickness computations;
- `Mask`: defines concepts, such as masks for blanking, defined in the Chimera technique;
- `Chim`: contains the grid assembly used within the Chimera technique;
- `Join`: deals with multi-block computations with various types of multi-block interface connectivity: 1-to-1 abutting, 1-to-n abutting or mismatched abutting.

The physical model layer includes the `Eos` module which computes quantities such as pressure, temperature (according to the equation of state of the gas) or laminar viscosity coefficient, and the `Tur` module already quoted above.

The space discretization layer gathers several important modules for the computation of the terms of the equations to solve and of the boundary conditions:

- `Oper`: defines the operator notion. Each operator class is responsible for the computation of a single term in CFD equations: convective flux (`Fxc`), diffusive flux (`Fxd`), source term (`Sou`);
- `Bnd`: contains the numerous classes devoted to the large variety of boundary conditions.

The solver layer contains:

- `Rhs`: builds the (explicit) right hand side of the equation system;
- `Lhs`: deals with all the implicit methods;
- `Tmo`: manages the main iterative time loop.

Finally, the top layer is the factory layer which is responsible of the dynamic creation of all objects (in OO methods, "objects" mean "instances of the class") and in particular includes the `Fact` module which implements several object "factories" to build objects from user input data coming from the interface.

In *elsA*, the choice of programming languages was done in order to allow both respect of the OO patterns of the design and of CPU constraints. The first objective led to the mandatory choice of a true OO programming language, and the C++ language was chosen, because it is the most widely used OO language, available on all usual platforms. Besides C++ as main language for implementing the OO design, it was decided to use Fortran for the two following reasons. First, the CPU tests we carried out at the beginning of the *elsA* project showed better CPU performance of Fortran in comparison with C. Second, some Fortran lines of legacy code were re-implemented in the *elsA* loops. However, due to a completely new design, it was in general not possible to keep the complete subroutines of the legacy code. These Fortran routines are very similar to private class methods: since they do not appear in the public class interface, they do not affect the OO design. A third programming language is used in *elsA*: the Python language which is a freely available interpreted OO language and is used for programming the *elsA* interface. Today, *elsA* includes more than one million lines (600,000 in C++ , 420,000 in Fortran and 55,000 in Python).

elsA interoperability

The interoperability of a program or of a piece of software is its ability to interact with another program or piece of software. The *elsA* software has evolved towards a software suite containing the *elsA* kernel (mainly the CFD solver) and some additional tools (dealing in particular with pre-processing and post-processing). All the elements of the suite should be seen as boxes in a larger simulation process. Our target is the integration of these boxes into all the platforms of our customers. Each aerospace industrial company already has a large set of programs and most of the time they have in-house software systems to manage them in their own process. Then our goal is rather to be able to be integrated than to integrate. The strategy for the interoperability and the so-called "component approach" software architecture we have designed for it are tightly bound to this goal of all-platforms integration.

The component approach

The *elsA* component approach is based on the interface description. We use standard interfaces for the CFD simulation components. The CGNS standard (see box 4) is preferred for the data model and the Python programming language (see the box describing this language in [10]) defines the protocol, in other words the order you should respect to use the component functions. Each box in the simulation workflow, including the *elsA* kernel, has to provide a Python interface supporting the CGNS data model: the CGNS/Python interface. When you integrate such a CGNS/Python component into a platform, the interface you use is not proprietary; it is based on Open System standards. A proprietary interface is a set of services which have a single implementation - the customer has no choice about the service provider - whilst an Open System interface should have more than one implementation. You can write Python scripts and just pass the CGNS data trees from one software tool to another in the memory of the process, or by means of a network if you have to, as long as they use the same public interface, no matter which implementation they use. This is the component interoperability which is the basis of Onera component approach.

CGNS/Python and *elsA*xdt

The CGNS/Python mapping of the CGNS/SIDS is used by an increasing number of Onera software components. Each tool has its own interface to CGNS/Python: the interface of the *elsA* kernel is *elsA*xdt. It parses CGNS/Python data trees and reads/writes *elsA* data in the tree for the next step of the workflow. We call this tree a “shuttle tree”, just like a shuttle bus going from code to code and picking up or dropping data. During this process, we avoid the use of files, because a file access often leads to problems on large computation clusters, and because most intermediate trees are sometimes not archived and trashed as soon as they are used. These transient trees can be created, used and killed within the memory.

A CGNS/Python computation with *elsA* uses quite simple commands, the read of the CGNS tree and the write of the resulting CGNS/Python tree:

```
import elsAxdt
parser=elsAxdt.XdtCGNS("001Disk.cgns")
parser.compute()
parser.save("Result.cgns")
```

In this simple example, all the required data is obviously embedded into the `001Disk.cgns` file: CGNS has been designed to be able to handle all the CFD data and even the specific solver parameters. Once the computation is performed, the output is obtained in `Result.cgns`, another CGNS tree.

Whereas in this first example we use files, the next example shows how a full memory transfer of CGNS/Python trees can be achieved:

```
import elsAxdt
from CGNS.MAP import *
(input_tree,links)=load("001Disk.cgns")
parser=elsAxdt.XdtPython(input_tree)
output_tree=parser.compute()
save("Result.cgns",output_tree,links)
```

The code lines are almost the same, but the architecture is quite different, because the *elsA* interface only uses the CGNS/Python tree. The actual load and save on the disk is not performed by *elsA*, it is performed by an Open Source module (`CGNS.MAP`) handling the CGNS/Python tree and HDF5 files.

We dissociate the *elsA* computation from the means used for the actual data exchange. You can use the CGNS/Python tree in memory for exchanges with a network layer or a dedicated proprietary database. For example, if you run a large parallel computation with local generation of grids, the *elsA* suite includes Python modules (Post, Converter, Generator) for this grid generation as CGNS/Python trees in memory. The solver runs on these grids, and produces one result per process. The merging of these trees is again performed in memory and the result is sent by means of a network to a remote post-processing workstation. Such a workflow limits the disk accesses on the high-performance computer and reduces the exchanges on the network to the data required for this particular application. This can be extended to the code coupling workflows [10].

Concluding remarks on interoperability

The *elsA*xdt interface is now used by some of our largest customers. These users are developing their own “integrated environment” with dedicated methods and tools. The common exchange backbone is based on CGNS. They can use the standard for file archiving as well as for component interoperability. The grid is generated as a CGNS tree by a commercial tool, then the proprietary process takes the tree and enriches it with *elsA* specific parameters, the tree is submitted to the solver and the result is directly passed to a commercial visualization tool.

The next step is now to use the CGNS/Python interface in the solver itself. Such a re-design is not necessary or useful for every part of the solver. We have selected a limited set of interfaces where we can explode the solver into separate and re-usable components. Each component would provide the CGNS/Python interface and thus would increase the interoperability with a finer granularity. The next solver generation would extend its flexibility up to its inner software components and thus would be a future platform for better research and better integration into the customers' proprietary platforms.

Box 4 - CGNS

The CGNS (CFD General Notation System) [24] provides a data model and a portable and extensible standard for exchanging and archiving of CFD analysis data. The main target is data associated with computed solutions of Navier-Stokes equations and their derivatives, but this can be applied to the field of computational physics in general.

CGNS started in the 1990s as a joint NASA, Boeing and McDonnell Douglas project. They developed the so-called SIDS (Standard Interface Data Structure) document that specifies the data model as a reference document. The first implementation was performed by ANSYS/ICEM teams on the top of the ADF (Advanced Data Format) proprietary low level storage system.

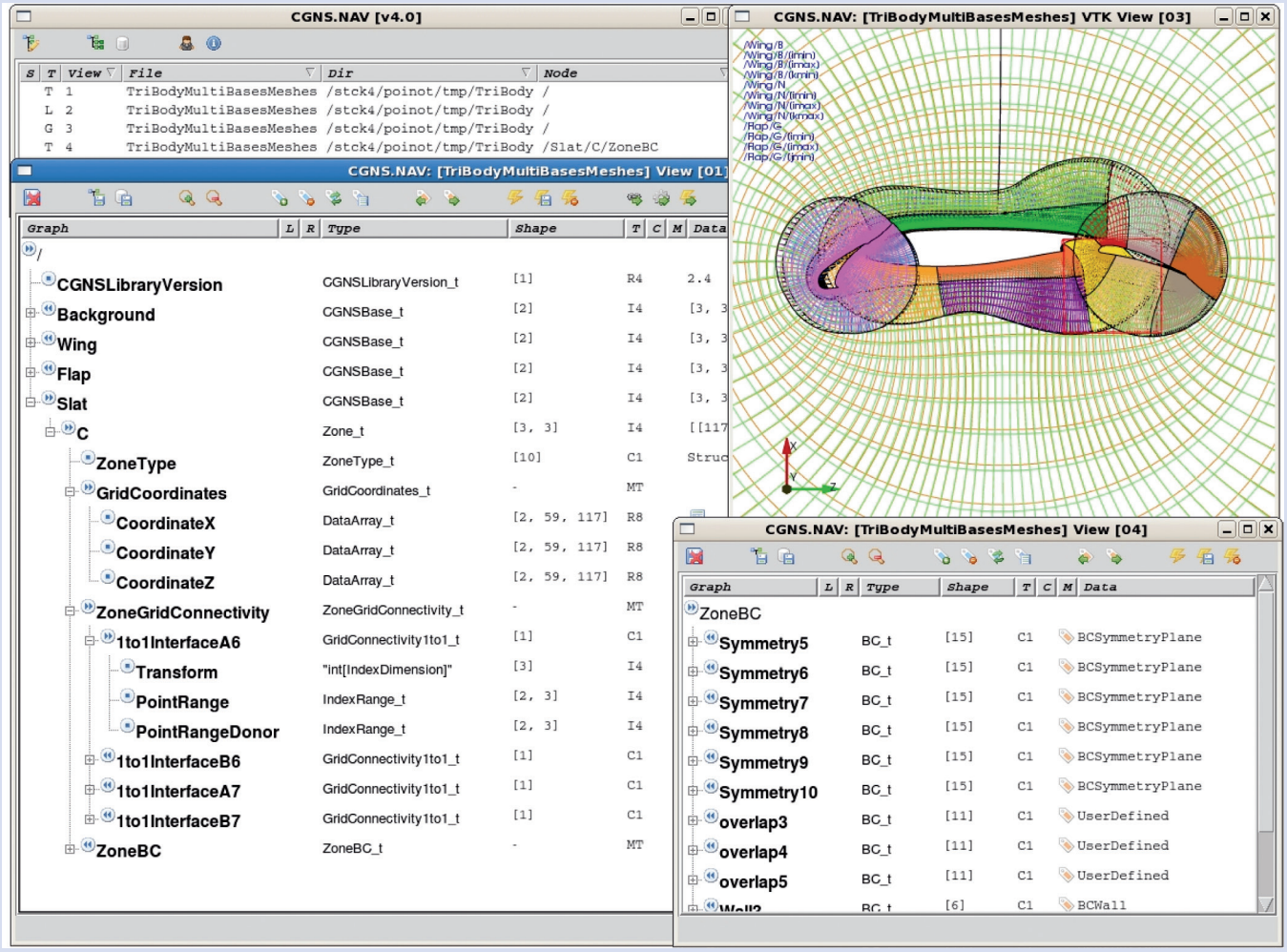


Figure B4-01 – Graphical view of a part of a CGNS tree using the Open Source CGNS.NAV tool

In 1999, the CGNS Steering Committee (CGNS/SC) was formed with academic organizations, aerospace industrial companies and software editors. Onera joined the CGNS/SC in 2001 and is an active member.

The standard is made up of a data model specification (CGNS/SIDS) which gives the name to the standard itself: Notation System. The main goal of the CGNS standard is to specify CFD data. The data can be used for exchange in a CFD workflow or as a storage conceptual model. Four CGNS implementations are available, the ADF, the HDF5 (Hierarchical Data File), the XML (eXtensible Markup Language) and finally the Python mapping. Now the main implementations are CGNS/HDF5 and CGNS/Python, the first is archiving oriented while the second is more workflow oriented. CGNS/Python is the basis of the interoperability system developed at Onera around *elsA*.

The standard is not used for internal data representation but only for the public view of the data, in a workflow exchange, for example from the CAD to the mesh tool, or from the solver to the visualizer. More complex workflows can be defined, with unsteady computation involving both CFD and CSM or optimization algorithms.

The data model has a tree structure, starting from the root node, the base, up to the smallest nodes that can be modeled, such as a single real value or a boundary condition application range (see figure B4-01).

The standard is extensible and the CGNS/SC has a process to add new structures when some user needs are raised and agreed by members. We often define new data structures in the framework of our projects. CGNS is the good candidate for data specification when several partners have to exchange data during run-time or to exchange computing files. CGNS can hold a complete simulation context and this has an important effect: when the user defines his data model, he has to find all the required and exact data needed for the simulation, this avoids hidden behavior of codes or unwanted side-effects.

Modeling capabilities

The *elsA* multi-application CFD simulation platform deals with internal and external aerodynamics from the low subsonic to the high supersonic flow regime and relies on the solving of the compressible 3-D Navier-Stokes equations (see Box 2). The thermodynamic properties of the fluid may correspond either to the perfect gas assumption or to the equilibrium real gas assumption described by a Mollier diagram. *elsA* allows the simulation of the flow around moving bodies in several formulations according to the use of absolute or relative velocities and the definition of the projection frame. This is useful when dealing with applications to turbomachinery flows, where the use of relative velocities is advantageous, and applications to flows around propellers, where the use of absolute velocities is necessary, with the same software. The bodies may be deformable, as well as the associated meshes. The gravity source term is optionally taken into account.

A large variety of turbulence models from eddy viscosity to full Differential Reynolds Stress models are implemented in *elsA* for the Reynolds averaged Navier-Stokes (RANS) equations (see [1] for further details). The range of turbulence models includes classical one-transport and two-transport equation models, more advanced two-equation models, multi-scale four-equation models, one-layer or two-layer Algebraic Reynolds Stress models. When thermal effects are important, specific versions of models are available. Low Reynolds versions of the models are mostly used for an accurate description of the boundary layer profiles, in association with the use of a very fine mesh near the wall. However, wall laws approximating the behavior of the boundary layer near the wall are also available, which allow for the use of coarser mesh near the walls and may be used to reduce the cost of the calculations, in particular for unsteady calculations.

Special attention has been paid to laminar-turbulent transition modeling [1], which may be a key point for obtaining accurate flow predictions. A reliable design of wings or turbines often requires an accurate modeling of the transition process. Transition prediction capability in the *elsA* RANS solver is based on application of criteria that either were previously developed at Onera for use in boundary layer codes, or result from classical criteria from literature. These criteria are used to describe Tollmen-Schlichting instabilities (including laminar separation bubble predictions), cross-flow instabilities, bypass for high external turbulence rate, attachment line contamination, wall roughness. This transition capability is available for complex geometry configurations. Nevertheless, to improve the applicability for very complex geometries, a transition model based on transport equations is now also available.

In order to deal with flows exhibiting strong unsteadiness and large separated regions, and/or to provide input data for aeroacoustic simulations, the user can perform Detached Eddy Simulations (DES) and Large Eddy Simulations (LES) [1]. The variants of DES methods available in *elsA* (basic DES, Zonal-DES, Delayed DES) are associated with the Spalart-Allmaras model or with the Menter (k, ω) model. The LES approach can be used to compute the larger structures of the turbulent flows while the smaller structures are dissipated by the numerical model, either by the MILES approach, which relies on the properties of advanced upwind schemes and dissipates the unresolved turbulent structures, or by subgrid models such as the Smagorinsky, Wale and filtered structure function models available in *elsA*.

Mesh topology capabilities

CFD solvers may rely on several meshing paradigms such as structured body-fitted grids, unstructured grids or structured Cartesian grids. Even when considering the same configuration, there is no universally accepted definitive choice, as can be seen by looking at the contributions to the Drag Prediction Workshop (http://aaac.larc.nasa.gov/tsab/cfdlarc/aiaa-dpw/Workshop4/presentations/DPW4_Presentations.htm). In that workshop, the contributions were roughly evenly divided, half of them on structured body-fitted grids and the other half on unstructured grids. Each of the mesh types has inherent advantages and disadvantages which may depend on the type of configuration and even more on the flow region of a given configuration. For example, Cartesian grids are easy to generate, to adapt, and to extend to higher-order spatial accuracy, but they are not suitable for resolving boundary layers around complex geometries. Body-fitted structured grids work well for resolving boundary layers, but the grid generation process for complex geometries remains tedious and requires considerable user expertise. General unstructured grids are well-suited to complex geometries and are relatively easy to generate, but their spatial accuracy is often limited to second order, and the associated data structures tend to be less computationally efficient than their structured-grid counterparts. Thus, today, the tendency is to couple meshing paradigms in the same software [29] or in the same coupling infrastructure [28], and this is also the way chosen for *elsA*.

In *elsA*, the main focus has been put firstly on structured body-fitted grids which allow for the use of very efficient numerical algorithms due to the natural (I, J, K) ordering of the hexahedral cells. Since it is generally impossible to define a unique structured body-fitted grid around complex geometries, the computational domain is divided in several adjacent or overlapping domains or blocks, in which simpler component grids can be generated more readily. Communication between component grids is achieved either by direct transfer or by interpolation across interfacing boundaries (patched grids), or by interpolation within overlapping grid regions (overset grids). In order to cope with more and more geometrically complex configurations, high flexibility advanced techniques of multi-block structured meshes are available in *elsA*, in addition to matching techniques for 1-to-1 abutting or 1-to-n abutting patched grids. These advanced matching techniques include quasi-conservative mismatched abutting patched grids (also called totally non-coincident matchings) and Chimera technique for overlapping meshes [8].

Mismatched abutting patched grids are intensively used in industry computations with *elsA*. The reason is that they simplify mesh generation for complex configurations, and reduce global number of mesh points for a given configuration, by preventing the propagation of mesh refinements throughout the computational domain. They are also well adapted to deal with sliding meshes.

The Chimera technique which enables a discretization of the flow equations in meshes composed of overset grids, may be applied to a wide range of configurations. The two main application domains of this method were originally dealing with the treatment of separate bodies (such as different positions of a missile below a wing) or with configurations including bodies in relative motion (for instance, helicopter rotor and fuselage or booster separation). During recent years, the Chimera method has also been increasingly used to simplify

and improve the meshing of complex configurations composed of the meshing of a basic geometry and of additional meshes adapted to joint bodies (for instance, a spoiler associated with a wing) or to geometrical details (for instance, technological effects in a turbomachinery row such as clearances, leakage slots, grooves, etc.). Finally, Chimera may also be used to achieve mesh adaptation for the simulation of phenomena involving a large range of length scales.

The development in *elsA* of hybrid multi-block capabilities allowing for the use of unstructured meshes in some blocks of a multi-block configuration [8] presently relies on a strong cooperative effort between Onera and Cerfacs. The objective is to benefit from the high flexibility of unstructured meshes in the flow regions where it becomes too difficult to build a structured grid. Tetrahedral, hexahedral, prismatic cells are considered in this *elsA* development, in order to be able to consider hexahedral cells near the walls, which is very important in aerodynamics for an accurate description of the boundary layers. A first set of hybrid capabilities will be available very soon in the main version of the *elsA* software. The mismatched abutting technique is extended in *elsA* to patch structured and unstructured grids. In the future, the plan is to also apply the Chimera technique to the matching between structured and unstructured grids.

Lastly, structured Cartesian grid capabilities are also currently being developed so that they can be part of the *elsA* suite in the future. The Cartesian formulation enables high order spatial discretization and mesh adaptation, which in turn allows for better capturing of off-body flow phenomena such as shear layers and wakes [8]. Since some specific high order schemes have been developed in this Cartesian solver, the way forward is to couple this solver with the block-structured solver of *elsA*. Matchings between blocks are again done using Chimera method. So, *elsA* will soon offer a quite complete multiple-gridding paradigm providing the potential for optimizing the gridding strategy on a local basis for the particular problem at hand, in order to cope with the increasing complexity of CFD applications (see [8] for further details on structured/unstructured and Cartesian/ curvilinear block matchings).

Numerics and boundary conditions capabilities

The flow equations are solved by a cell centered finite-volume method [8]. Space discretization schemes include a range of second order centered or upwind schemes. Centered schemes are stabilized by scalar or matrix artificial dissipation, including damping capabilities inside viscous layers, in order to preserve accuracy. Upwind schemes are based on numerical fluxes such as van Leer, Roe, Coquel-Liou HUS, or AUSM fluxes and are associated with classical slope limiters. Second and third order Residual Based Compact schemes are also available in *elsA*. The semi-discrete equations are integrated, either by multistage Runge-Kutta schemes with implicit residual smoothing, or by backward Euler integration with implicit schemes solved by robust LU relaxation methods [21], which in general leads to a higher efficiency with *elsA*. An efficient multigrid technique can be selected in order to accelerate convergence. For time accurate computations, the implicit dual time stepping method or the Gear integration scheme are employed. Preconditioning is used for low speed flow simulations.

An extensive range of boundary conditions is available in *elsA*, from standard inlet, outlet or wall conditions, to more specific conditions for helicopter configurations (such as the so-called “Froude” far field boundary conditions in hover) or turbomachinery configurations (such as the radial equilibrium condition, or – see Box 5 – the Reduced Blade Count method and the phase-lagged technique for the simulation of rotor/stator interactions). Actuator-disc models are available to economically model the effects of helicopter rotors or propellers, when complete detailed calculations are not worthwhile.

Box 5 - Reduced Blade Count method and Phase-Lagged technique, two different approaches for simulating the time-periodic flow in a turbomachinery stage configuration

To improve turbomachinery performances, 3D Navier-Stokes flow computations in blade rows are commonly used for turbine and compressor design. Approximate steady flow calculations through multi-stage machines have become usual in design process for many years. In *elsA*, they are performed using a specific steady condition, the mixing plane condition, to connect two consecutive rows. This condition is based on azimuthal averages which are computed at the interfaces and transferred from a row to the consecutive one. It gives a quite good prediction of the overall efficiency of a machine but of course does not give any information on the unsteady flow fluctuations. Unsteady computations are increasingly used for industrial purposes: in a complete staged machine, they are necessary when performing unsteady non periodic phenomena. They are still very expensive, as discussed in the High Performance Computing section of this paper. But under some conditions, techniques for reducing the computational domain can be used. They can be applied for time-periodic flows (in the frame of reference of each row), that is for flows where the unsteadiness is only due to the relative motion of the rows. The first technique, called “Reduced Blade Count” method [11], was introduced in *elsA* software [23]. The computation is performed on the actual geometry with reduced blade counts. The interface between the blade rows accounts for the non equal pitches on each side of the interface by an appropriate scaling. The second technique, known as “Phase-Lagged” technique [9], or “Chorochronic” method was implemented in 2003 in *elsA* software [2]. A single blade passage is computed for each row. The flow solution is stored on the interface boundaries and on the azimuthal periodic boundaries to deal with the phase lag which exists between rows and adjacent blade passages in a row.

Let us note N_1 and N_2 the actual blade numbers of two consecutive rows, ω_1 and ω_2 their rotation speeds. The flow periods are equal to $T_1=T_{rot}/N_2$ and $T_2=T_{rot}/N_1$, respectively in the rotating frame of the first row and in the rotating frame of the second one, $T_{rot}=2\pi/|\omega_2-\omega_1|$ being the time for a blade passage to make a whole revolution.

The "Reduced Blade Count" technique consists in reducing the computational domain to K_1 and K_2 blade passages without changing the geometry. K_1 and K_2 are chosen such that K_1 / N_1 and K_2 / N_2 are about the same. We assume that the flow is identical on two consecutive K_i blade passages at each instant, which is exact when the ratios K_i / N_i are equal. Between the lower and the upper boundaries, the flow continuity is enforced, which is an approximation with respect to the actual flow, since the real phase lag between these boundaries is cancelled.

We can define D_M as a mean value of the K_i / N_i ratios. This quantity represents the geometric reduction which is applied to each row to obtain the computational domain. On the interface, the two blade groups are linked by an instantaneous continuity condition through a common azimuthal extension $e_m = 2\pi / D_M$ with a scaling given by $\lambda_i = (N_i / K_i)\ (1/D_M)$ for each row. The "Reduced Blade Count" technique induces a slight approximation since the computational periods are $T'_1 = T_{rot} / (K_2 D_M)$ and $T'_2 = T_{rot} / (K_1 D_M)$, that is $T'_i = \lambda_i T_i$. This technique can be applied to several consecutive rows if the actual blade numbers are cooperative enough to apply a geometric reduction.

In the "Phase-Lagged" approach, the computational domain is limited to a single blade passage for each row. As the flow is time-periodic in each blade row, a phase lag exists between two adjacent blade passages. This phase lag is the time taken by a blade of the next row to cover the pitch of the row, modulo the time period of the row. The "Phase-Lagged" technique consists of storing the flow values on the azimuthal periodic boundaries and on the interface in order to use them later to build the flow.

Let us consider two space periodic points A and B of the upper and lower boundaries of the first row. B is ahead of A and what happens at time t in B will happen in A at $t + T_2$ and more generally at $t + T_2 + nT_1$ or has happened at $t + T_2 - mT_1$ (m is an integer such that $T_2 - mT_1$ is negative). The flow condition stored in A at time $t + T_2 - mT_1$ can be used for the boundary treatment of B at time t.

The treatment of the interface relies on the same principle. At each time step, the flow continuity is enforced at the interface between one cell facet of the first row and the suitable storage of the second row, taking into account for the relative position of the rows and using the necessary spatial interpolation on the stored data.

The direct storage of flow solution may lead to very large requirements in terms of memory. The data storage is lowered to an acceptable amount by Fourier analysis.

In the framework of the TATEF2 European project [7], unsteady flow simulations of the stator-rotor interaction in a transonic turbine stage have been performed using the "Phase-lagged" approach. Figure B5-01 shows clearly shock structures obtained for high pressure ratio.

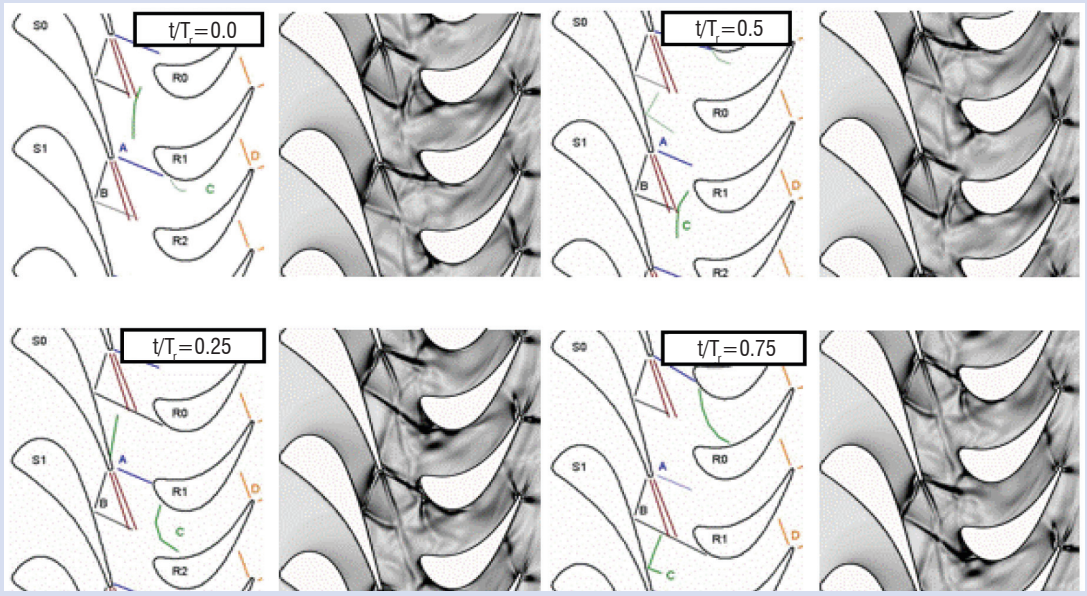


Figure B5-01 – Shock progression through the turbomachinery stage: schematic (left) and density gradient (right). $y/H \sim 25\%$.

The "Phase-Lagged" approach does not make any approximation on the number of actual blades and it accounts for the real time period in each reference frame. So it is more accurate than the "Reduced Blade Count" technique. Moreover, as the computational domain is limited to a single blade passage for each row, it is less expensive in terms of CPU time and computer memory. But this approach can only be applied to a single stage, whereas the "Reduced Blade Count" technique can be applied to multi-stage configurations, even when several rotors have different rotational speeds.

A generalization of the "Phase-Lagged" approach, called "multiple frequency Phase-Lagged method", is under development in *elsA* [18]. This method allows for unsteady computations through several rows, whilst still limiting the computational domain to one single blade-to-blade passage in each row.

Multidisciplinary and optimization capabilities

elsA also includes the `Ael` module offering a general framework for aeroelastic applications [10]. This module provides for the following simulations:

- harmonic forced motion simulations for a given structural mode;
- linearized Euler or Navier-Stokes simulations;
- static and dynamic fluid/structure coupling simulations in time domain with different levels of structural modeling ("reduced flexibility matrix" approach for static coupling, modal approach, full finite element structural model).

The `Opt` module dealing with calculation of sensitivities by linearized equation or by adjoint solver techniques is useful for optimization and control [22]. The calculation of sensitivities consists in calculating the derivatives with respect to control parameters of objective functions such as drag or lift.

High Performance Computing: towards massively parallel computations

As said above, typical aeronautic configurations nowadays take into account fine geometrical details, resulting in huge problem sizes, as the example of the simulation of the complete multistage compressor in Figure 4 (a simulation which does not use the approximate techniques described in Box 5 for reducing the computational domain). Moreover, although the type of physical modeling still remains mostly RANS, the major trend is to move towards URANS and even DES or LES in order to get more accurate results, leading to an estimated additional CPU cost of about two orders of magnitude.

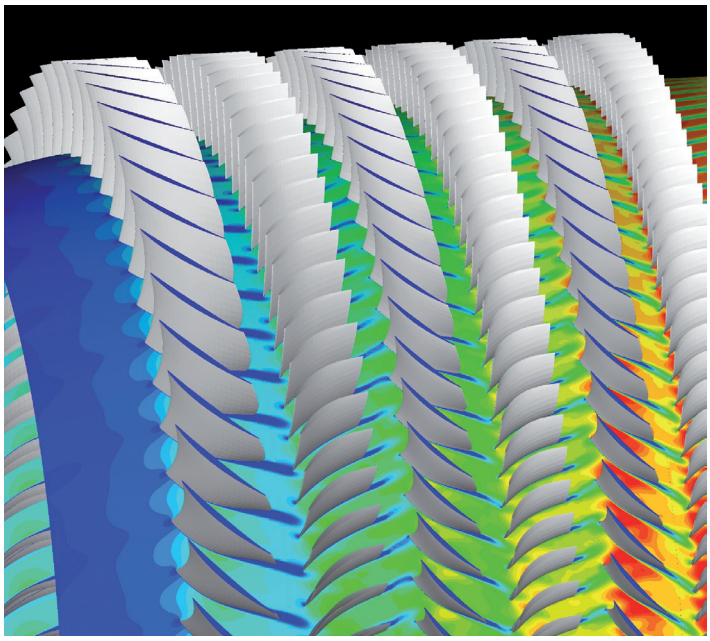
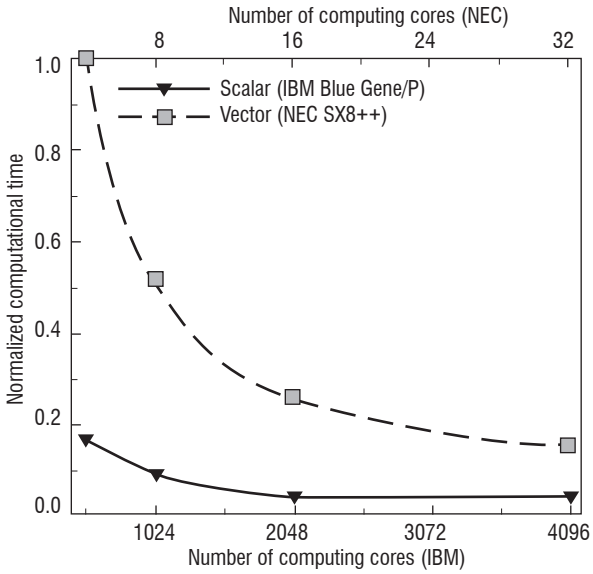
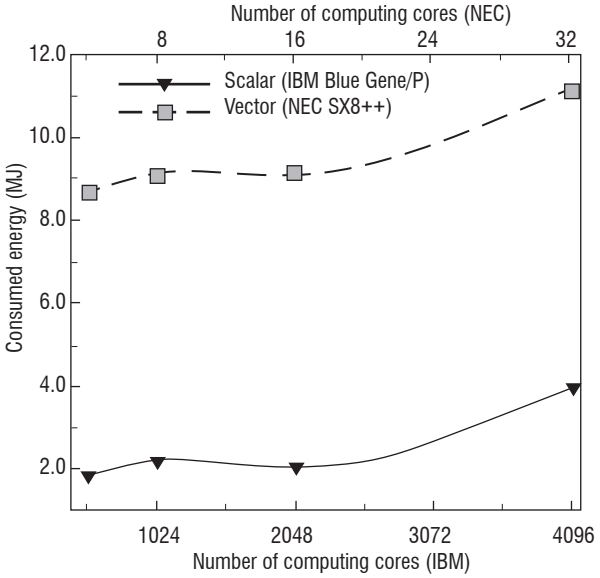


Figure 4 – Unsteady flow simulation of a multistage compressor (134 million cells) using 512 to 4096 computing cores (SNECMA configuration) [17]

To handle such demands, High Performance Computing is now unavoidable in order first to tackle simulations with very large number of points (and thus requiring a huge amount of memory), and second to reduce the CPU wall clock time as much as possible. Onera, Cerfacs and CS have made very large efforts [15, 16] to improve the code's performance on state-of-the-art vector and x86-64 based computing platforms. In the CFD context, vector machines are now being rapidly supplanted by clusters of x86-64 based nodes due to their high operating cost and relatively poor energy efficiency (see in Figure 5 comparison between many-core and vector platforms on the simulation presented in Figure 4). *elsA* performance improvement efforts are now entirely dedicated to massively parallel computers.



a) CPU time (from [17])



b) Consumed energy (from [17])

Figure 5 – Comparison between many-core and vector platforms for a single unsteady physical iteration on the multistage compressor configuration

Parallel strategy for structured multi-block calculations with *elsA*

The Message Passing Interface (MPI) standard library is used to implement communications between processors. *elsA* uses a standard coarse-grained SPMD approach: each block is allocated to a processor. Several blocks can be allocated to the same processor.

As said previously, meshing a configuration with structured grids often leads to a complex multi-block topology, thus requiring specific treatments to exchange data between adjacent blocks. In the *elsA* solver, each block is surrounded by two layers of ghost cells storing values coming from the adjacent block. If the two blocks are allocated to two different computing cores, then point-to-point message passing communication occurs. Otherwise, ghost cells are directly filled by a memory-to-memory copy.

Point-to-point communications are implemented either with blocking (`MPI_Sendrecv_replace`) or non-blocking (`MPI_Irecv/MPI_send`) point-to-point messages. Only blocking point-to-point communications require the scheduling of messages as shown by Fig. 6. The scheduling of communications comes from a heuristic coloring algorithm adapted from graph theory.

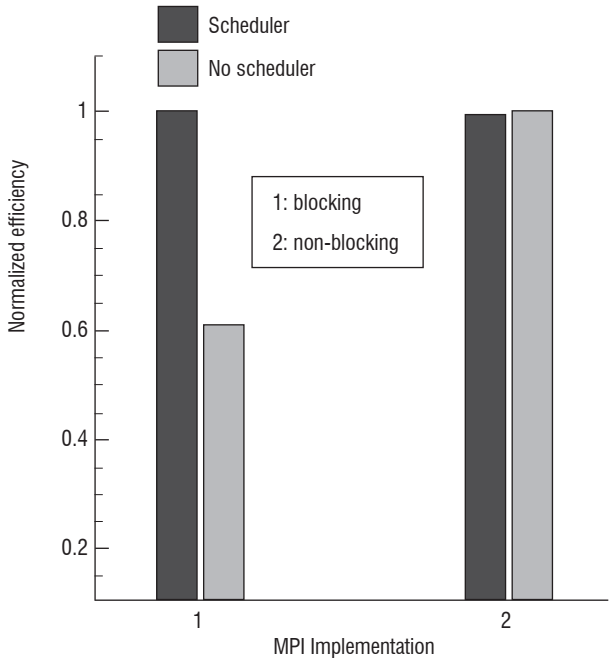


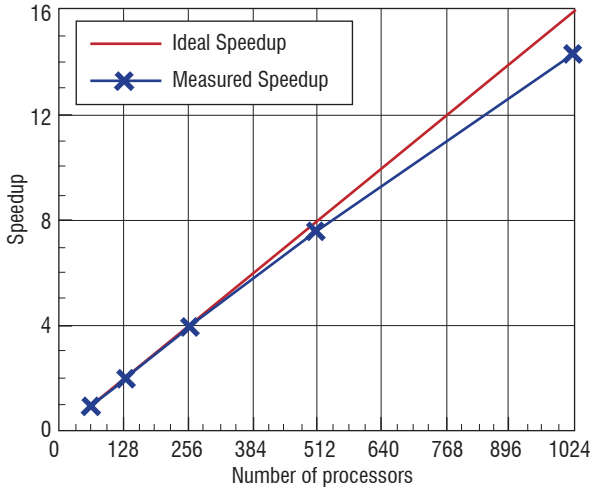
Figure 6 – Effect of the MPI blocking and non-blocking communications on *elsA* efficiency with and without message scheduling

The non-coincident inter-block connectivity is implemented with blocking collective communications (`MPI_Allgatherv`). If only one computing core handles the whole inter-block connection, data exchanges come down to memory-to-memory copies without MPI messages.

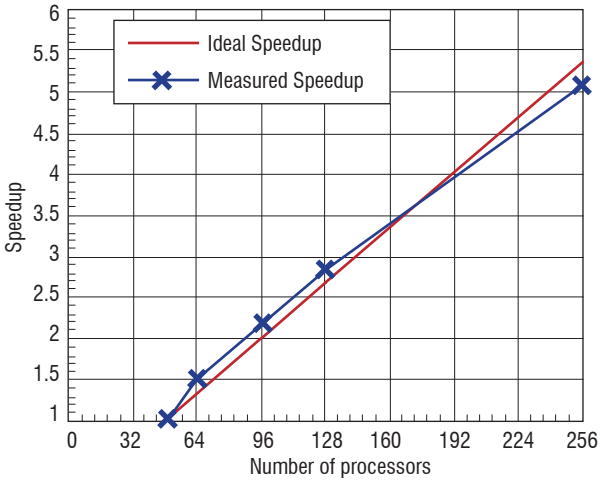
Performance discussion

The well known Amdahl's law states that load balancing is crucial to obtain a good efficiency on parallel computers when the number of processors increases. Due to topological constraints, the number of cells in blocks can be very different, ranging from 10^3 to 100^3. Most of the time, some blocks must be split to achieve a good load balancing. It is not clear if an optimal response can be obtained in a reasonable amount of time. Therefore, *elsA* integrates a heuristic block splitting algorithm in the load balancing process. Based on the relative error between the number of cells allocated to the computing core and the ideal number of cells, it checks if the largest block to be allocated needs to be split. The partitioning algorithm handles many constraints such as the multigrid constraint. The so-called "greedy" load balancing algorithm loops over all of the blocks searching for the largest one in terms of cells and allocates it to the computing core with the fewest cells until all of the blocks are allocated. Note that the number of ghost cells increases with the number of blocks split, leading to an increasing problem size. Topology modification also implies carefully handling block-based implicit algorithms since convergence may rapidly be degraded. Therefore, communications occur at each relaxation step inside the implicit LU stage.

elsA has been ported to most high-performance computing platforms, achieving good CPU efficiency on both scalar multi-core computers and vector computers. As an example, Figure 7a shows typical speedup results on a civil aircraft configuration including 27,8 10^6 mesh points and 1037 blocks. The numerical options include multigrid algorithm (3 levels) and the Spalart-Allmaras one-equation turbulence model. For large number of processors, the configuration has been split, ending up with 1774 blocks. The computer is the Cerfacs' BlueGene/L computer. Another concrete example (Figure 7b) is the High Lift Prediction Workshop configuration, where a good scalability is obtained up to 256 computing cores, on a SGI cluster built with Intel Nehalem processors (Onera's Stelvio computer), on a grid of 160 10^6 mesh points, in 1235 blocks.



a) Generic civil aircraft configuration



b) Configuration of the High Lift Prediction Workshop 2010

Figure 7 – Normalized speedup obtained with *elsA* on two configurations

Future challenges in HPC

Since the performance improvement through increases in clock frequency has reached a limit, mainly due to the power dissipation problem, new parallel paradigms have emerged, namely the increase of the number of cores on a chip and the use of specialized accelerators (FPGA, GPU, Cell processor). To handle such paradigms, we have gained experience in two approaches: OpenMP thread programming and GPU using CUDA. In our previous studies on thread parallelism [20], OpenMP appeared to be less efficient compared with MPI on nodes with small numbers of cores (<16). We plan to update these studies in the context of upcoming many-core chips. In addition, we are currently building a prototype version dedicated to GPU; here the main challenge is the limited data transfer bandwidth between CPU host and GPU.

As memory bandwidth will probably remain the limiting factor on performance in the near future, major efforts are planned for both the fine-grain parallelism (data-parallelism) and the optimized use of cache memory. This will be mandatory as the trend in increasing the number of cores implies that many cores will compete for hardware resources.

To summarize on parallel performance, *elsA* is a portable code reasonably well adapted to current generations of HPC platforms and continuous work is undertaken to strike a balance between good efficiency and maintainability. *elsA* is routinely used on hundreds of processors in industry; the biggest computation was run with 8192 cores on a 1.7 10^9 point grid [12].

Note also that not only the solver, but also the pre- and post-processing steps should be fully and efficiently addressed for massively parallel configurations. Moreover, the efficiency of the whole simulation in the framework of a multiple-gridding paradigm will be a big challenge for the next few years ■

Acknowledgements

The authors would like to thank all colleagues of several Onera Departments (CFD and Aeroacoustics, Applied Aerodynamics, Aerodynamics and Energetics Modeling, Aeroelasticity and Structural Dynamics), of Cerfacs and of development partners for their sustained effort to make the *elsA* software a powerful and reliable tool for research and for industry.

We would like in particular to acknowledge contributions of the following former and present members of the *elsA* software team in the Onera CFD and Aeroacoustics Department: A. Gazaix-Jollès, M. Lazareff, J. Mayeur, B. Michel, P. Raud, A.-M. Vuillot.

We also thank our colleagues who have provided us with the *elsA* results presented in the paper: F. Blanc (Cerfacs/Airbus) for the transport aircraft configuration, N. Gourdain (Onera Applied Aerodynamics Department, then Cerfacs) for the rotating stall turbomachinery simulation, then for the HPC turbomachinery calculation, G. Delattre (Onera Applied Aerodynamics Department) for the CROR configuration, C. Benoit, G. Jeanfaivre and X. Juvigny (Onera CFD and Aeroacoustics Department) for the helicopter configuration.

References

[1] B. AUPOIX, D. ARNAL, H. BÉZARD, B. CHAOUAT, F. CHEDEVERGNE, S. DECK, V. GLEIZE, P. GRENARD and E. LAROCHE - *Transition and Turbulence Modeling*. Aerospace Lab, Issue 2, 2011.

[2] G. BILLONNET, S. PLOT and G. LEROY - *Implementation of the elsA Software for the Industrial Needs of Flow Computations in Centrifugal Compressors*. 41e colloque d'Aérodynamique Appliquée, Lyon, 2006.

[3] R.H. BUSH, G.D. POWER and C.E. TOWNE - *WIND: The Production Flow Solver of the NPARC Alliance.* AIAA Paper 98-0935, 36th AIAA Aerospace Science Meeting and Exhibit, Reno, 1998.

[4] L. CAMBIER, M. GAZAIX - *elsA: an Efficient Object-Oriented Solution to CFD Complexity.* AIAA Paper 2002-0108, 40th AIAA Aerospace Science Meeting and Exhibit, Reno, 2002.

[5] L. CAMBIER and N. KROLL - *MIRACLE - A joint DLR/Onera Effort on Harmonization and Development of Industrial and Research Aerodynamic Computational Environment*. Aerospace Science and Technology, vol.12, 2008.

[6] L. CAMBIER and J.-P. VEUILLOT - *Status of the elsA CFD Software for Flow Simulation and Multidisciplinary Applications*. AIAA Paper 2008-664, 46th AIAA Aerospace Science Meeting, Reno, 2008.

[7] L. CASTILLON, G. PANIAGUA, T. YASA, A. DE LA LOMA and T. COTON - *Unsteady Strong Shock Interactions in a Transonic Turbine: Experimental and Numerical Analysis*. ISABE Paper 2007-1218, Beijing, China, 2007.

[8] B. COURBET, C. BENOIT, V. COUAILLIER, F. HAIDER, M.-C. LE PAPE and S. PÉRON - *Space Discretization Methods*. Aerospace Lab, Issue 2, 2011.

[9] J.I. ERDOS and E. ALZNER - *Computation of Unsteady Transonic Flows Through Rotating and Stationary Cascades*. NASA CR-2900, 1977.

[10] M. ERRERA, A. DUGEAI, P. GIRODROUX-LAVIGNE, J.-D. GARAUD, M. POINOT, S. CERQUEIRA and G. CHAINERAY - *Multi-Physics Coupling Approaches for Aerospace Numerical Simulations*. Aerospace Lab, Issue 2, 2011.

[11] A. FOURMAUX - *Assessment of a Low Storage Technique for Multi-Stage Turbomachinery Navier-Stokes Computations*. ASME Winter Annual Meeting, Chicago, 1994.

[12] M. GAZAIX and S. CHAMPAGNEUX - *Recent Results with elsA on Multi-Cores*. Symposium on CFD on Future Architectures, DLR Braunschweig, 2009.

[13] M. GAZAIX, A. JOLLÈS and M. LAZAREFF - *The elsA Object-Oriented Tool for Industrial Applications*. 23rd ICAS meeting, Toronto, 2002.

[14] T. GERHOLD, V. HANNEMANN and D. SCHWAMBORN - *On the Validation of the DLR-TAU Code*. in W. NITSCHE, H.J. HEINEMANN and R. HILBIG (Eds), New Results in Numerical and Experimental Fluid Mechanics, NNFM 72, Vieweg, 1999.

[15] N. GOURDAIN, L.Y.M. GICQUEL, M. MONTAGNAC, O. VERMOREL, M. GAZAIX, G. STAFFELBACH, M. GARCIA, J.-F. BOUSSUGE and T. POINSOT - *High Performance Parallel Computing of Flows in Complex Geometries - Part 1: Methods*. Computational Science and Discovery, vol. 2, 2009.

[16] N. GOURDAIN, L.Y.M. GICQUEL, G. STAFFELBACH, O. VERMOREL, F. DUCHAINE, J.-F. BOUSSUGE and T. POINSOT - *High Performance Parallel Computing of Flows in Complex Geometries - Part 2: Applications*. Computational Science and Discovery, vol. 2, 2009.
[17] N. GOURDAIN, M. MONTAGNAC, F. WLASSOW and M. GAZAIX - *High-performance Computing to Simulate Large-scale Industrial Flows in Multistage Compressors*. International Journal of High Performance Computing Applications, vol. 24, 2010.
[18] L. HE and H.D. LI - *Single-Passage Analysis of Unsteady Flows Around Vibrating Blades of a Transonic Fan Under Inlet Distortion.* Journal of Turbomachinery, 2002.
[19] W.L. KLEB, E.J. NIELSEN, P.A. GNOFFO, M.A. PARK and W.A. WOOD - *Collaborative Software Development in Support of Fast Adaptive AeroSpace Tools (FAAST)*. AIAA Paper 2003-3978, 33rd AIAA Fluid Dynamics Conference, Orlando, 2003.
[20] F. LOERCHER - *Optimisation d'un code numérique pour des machines scalaires et parallélisation avec OpenMP et MPI.* Technical report TR/CFD/03/104, CERFACS, 2003.
[21] C. MARMIGNON, V. COUAILLIER and B. COURBET - *Solution Strategies for Integration of Semi-Discretized Flow Equations in elsA and CEDRE.* Aerospace Lab, Issue 2, 2011.
[22] J. PETER, G. CARRIER, D. BAILLY, P. KLOTZ, M. MARCELET and F. RENAC - *Local and Global Search Methods for Design in Aeronautics*. Aerospace Lab, Issue 2, 2011.
[23] S. PLOT, G. BILLONNET and L. CASTILLON - *Turbomachinery Flow Simulations Using elsA Software: Steady Validations and First Abilities for Unsteady Computations*. 38th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, Indianapolis, 2002.
[24] D. POIRIER, S.R. ALLMARAS, D.R. McCARTHY, M.F. SMITH and F.Y. ENOMOTO - *The CGNS system.* AIAA Paper 1998-3007, 29th AIAA Fluid Dynamics Conference, Albuquerque, 1998.
[25] J. RENEAUX, P. BEAUMIER and P. GIRODROUX-LAVIGNE - *Advanced Aerodynamic Applications with the elsA Software*. Aerospace Lab, Issue 2, 2011.
[26] C.C. ROSSOW and L. CAMBIER - *European Numerical Aerodynamics Simulation Systems.* in E.H. HIRSCHEL and E. KRAUSE (Eds.), *100 Volumes of 'Notes on Numerical Fluid Mechanics'*, NNFM 100, Springer-Verlag, 2009.
[27] J.B. VOS, A. RIZZI, D. DARRACQ and E.H. HIRSCHEL - *Navier-Stokes Solvers in European Aircraft Design.* Progress in Aerospace Sciences 38, 2002.
[28] A.M. WISSINK, J. SITARAMAN, V. SANKARAN, D.J. MAVRIPLIS and T.H. PULLIAM - *A Multi-code Python-Based Infrastructure for Overset CFD with Adaptive Cartesian Grids*. AIAA Paper 2008-927, 46th AIAA Aerospace Science Meeting, Reno, 2008.
[29] H. YANG, D. NUERNBERGER and H.-P. KERSKEN - *Towards Excellence in Turbomachinery CFD: a Hybrid Structured-Unstructured RANS Solver.* Journal of Turbomachinery, vol. 128, 2006.

Acronyms

ADF (Advanced Data Format)
AUSM (Advection Upstream Splitting Method)
CAD (Computer-Aided Design)
CFD (Computational Fluid Dynamics)
CGNS (CFD General Notation System)
CROR (Counter Rotating Open Rotor)
CSM (Computational Structural Mechanics)
CUDA (Compute Unified Device Architecture)
DES (Detached Eddy Simulation)
elsA (ensemble logiciel pour la simulation en Aérodynamique)
FPGA (Field-Programmable Gate Array)
GPU (Graphics Processing Unit)
HDF (Hierarchical Data File)
HPC (High Performance Computing)
HUS (Hybrid Upwind Splitting)
LES (Large Eddy Simulation)
LU (Lower Upper)
MPI (Message Passing Interface)
OO (Object-Oriented)
RANS (Reynolds Averaged Navier-Stokes)
SIDS (Standard Interface Data Structure)
SPMD (Single Process, Multiple Data)
TATEF2 (Turbine Aero-Thermal External Flows 2)
UML (Unified Modeling Language)
URANS (Unsteady Reynolds Averaged Navier-Stokes)
XML (eXtensible Markup Language)

Laurent Cambier graduated in 1980 from "École Centrale de Paris", is presently Assistant Director of the CFD and Aeroacoustics department of Onera and Head of the *elsA* program coordinating research, software and validation activities.

Michel Gazaix graduated in 1979 from "École Normale Supérieure de Saint-Cloud" and has been a Research Scientist at Onera since 1986. In the CFD and Aeroacoustics department, his research topics include: High Performance Computing, Real Gas modeling in compressible flows, and software engineering applied to large scientific codes.

Sébastien Heib PhD in Numerical Analysis from Paris 6 University, joined Onera in 2001. He is presently the head of the *elsA* software project in the CFD and Aeroacoustics department.

Sylvie Plot graduated from "École Nationale Supérieure de l'Aéronautique et de l'Espace" in 1990. Since that time, she has been working at Onera, mainly on the development of helicopter and turbomachinery CFD capabilities. She is currently head of the "Software and Advanced Simulations for Aerodynamics" research unit of the CFD and Aeroacoustics Department.

Marc Poinot who received an MSc in Computer Science from Paris XI Orsay University, is a software engineer in the CFD and Aeroacoustics department of Onera. He is in charge of interoperability topics for code coupling and numerical simulation platforms. He is member of the CGNS steering committee and he initiated and actively participated to the migration of CGNS to HDF5.

Jean-Pierre Veuillot graduated in 1969 from "École Supérieure de l'Aéronautique", received a PhD from Paris 6 University in 1975. He was Assistant Director of the CFD and Aeroacoustics department when he retired from Onera in 2010 after a whole career at Onera devoted to numerical methods and software development in CFD.

Jean-François Boussuge graduated from the Von Karman Institute, was working as numerical simulation engineer in the automotive field until 2001 before becoming research engineer at CERFACS where he was in charge of code development on various CFD methods and algorithms until 2005. Today, he is the leader of the external aerodynamics group at CERFACS.

Marc Montagnac is a research engineer in external aerodynamics group at CERFACS. He graduated from the "Institut National des Sciences Appliquées" with an engineering diploma in 1994 and received a PhD in Computer Science from Paris 6 University in 1999. His primary research and professional interests are in the areas of software engineering, High Performance Computing and numerical methods.