RESEARCH CENTRE

Inria Centre at Université Côte d'Azur

2024

ACTIVITY REPORT

Project-Team

ECUADOR

Program transformations for scientific computing

DOMAIN

Applied Mathematics, Computation and Simulation

THEME

Numerical schemes and simulations

Inria

Contents

Project-Team ECUADOR

Creation of the Project-Team: 2014 January 01

Keywords

Computer sciences and digital sciences

A2.1.1. – Semantics of programming languages

A2.2.1. – Static analysis

A2.5. – Software engineering

A6.1.1. – Continuous Modeling (PDE, ODE)

A6.2.6. – Optimization

A6.2.7. – High performance computing

A6.3.1. – Inverse problems

A6.3.2. – Data assimilation

Other research topics and application domains

B1.1.2. – Molecular and cellular biology

B3.2. – Climate and meteorology

B3.3.2. – Water: sea & ocean, lake & river

B3.3.4. – Atmosphere

B5.2.3. – Aviation

B5.2.4. – Aerospace

B9.6.3. – Economy, Finance

1 Team members, visitors, external collaborators

Research Scientists

- Laurent Hascoët [Team leader, INRIA, Senior Researcher]
- Jean Luc Bouchot [INRIA, Advanced Research Position]
- Alain Dervieux [INRIA, Emeritus]

PhD Student

- Bastien Sauvage [INRIA, until Sep 2024]

Administrative Assistant

- Christine Claux [INRIA]

Visiting Scientists

- Ines Naumann [KEMPEN KRAUSE INGENIEURE GmbH, from May 2024 until Jun 2024]
- Uwe Naumann [Univ RWTH AACHEN, from May 2024 until Jun 2024]

External Collaborators

- Bruno Koobus [IMAG, University of Montpellier]
- Stephen Wornom [IMAG, University of Montpellier]

2 Overall objectives

Team Ecuador studies Algorithmic Differentiation (AD) of computer programs, blending :

- **AD theory:** We study software engineering techniques, to analyze and transform programs mechanically. Algorithmic Differentiation (AD) transforms a program `P` that computes a function F, into a program `P'` that computes analytical derivatives of F. We put emphasis on the *adjoint mode* of AD, a sophisticated transformation that yields gradients for optimization at a remarkably low cost.
- **AD application to Scientific Computing:** We adapt the strategies of Scientific Computing to take full advantage of AD. We validate our work on real-size applications.

We aim to produce AD code that can compete with hand-written sensitivity and adjoint programs used in the industry. We implement our algorithms into the tool Tapenade, one of the most popular AD tools at present.

Our research directions :

- Efficient adjoint AD of frequent dialects e.g. Fixed-Point loops.
- Development of the adjoint AD model towards Dynamic Memory Management.
- Evolution of the adjoint AD model to keep in pace with modern programming languages constructs.
- Optimal shape design and optimal control for steady and unsteady simulations. Higher-order derivatives for uncertainty quantification.
- Adjoint-driven mesh adaptation.

3 Research program

3.1 Algorithmic Differentiation

Participants: Laurent Hascoët, Jean-Luc Bouchot.

Glossary

algorithmic differentiation (AD, aka Automatic Differentiation) Transformation of a program, that returns a new program that computes derivatives of the initial program, i.e. some combination of the partial derivatives of the program's outputs with respect to its inputs.

adjoint Mathematical manipulation of the Partial Differential Equations that define a problem, obtaining new differential equations that define the gradient of the original problem's solution.

checkpointing General trade-off technique, used in adjoint AD, that trades duplicate execution of a part of the program to save some memory space that was used to save intermediate results.

Algorithmic Differentiation (AD) differentiates *programs*. The input of AD is a source program P that, given some $X \in \mathbb{R}^n$, returns some $Y = F(X) \in \mathbb{R}^m$, for a differentiable F. AD generates a new source program P' that, given X, computes some derivatives of F [4].

Any execution of P amounts to a sequence of instructions, which is identified with a composition of vector functions. Thus, if

$$\begin{array}{lll} P & \text{runs} & \{I_1; I_2; \dots I_p;\}, \\ F & \text{then is} & f_p \circ f_{p-1} \circ \dots \circ f_1, \end{array} \tag{1}$$

where each f_k is the elementary function implemented by instruction I_k. AD applies the chain rule to obtain derivatives of F. Calling X_k the values of all variables after instruction I_k, i.e. $X_0 = X$ and $X_k = f_k(X_{k-1})$, the Jacobian of F is

$$F'(X) = f'_p(X_{p-1}) \,.\, f'_{p-1}(X_{p-2}) \,.\, \dots \,.\, f'_1(X_0) \tag{2}$$

which can be mechanically written as a sequence of instructions I'_k. This can be generalized to higher level derivatives, Taylor series, etc. Combining the I'_k with the control of P yields P', and therefore this differentiation is piecewise.

The above computation of $F'(X)$, albeit simple and mechanical, can be prohibitively expensive on large codes. In practice, many applications only need cheaper projections of $F'(X)$ such as:

- **Sensitivities**, defined for a given direction $\dot{X}$ in the input space as:

$$F'(X).\dot{X} = f'_p(X_{p-1}) \,.\, f'_{p-1}(X_{p-2}) \,.\, \dots \,.\, f'_1(X_0) \,.\, \dot{X} \quad . \tag{3}$$

  This expression is easily computed from right to left, interleaved with the original program instructions. This is the *tangent mode* of AD.

- **Adjoints**, defined after transposition (F'^*), for a given weighting $\overline{Y}$ of the outputs as:

$$F'^*(X).\overline{Y} = f'^*_1(X_0).f'^*_2(X_1).\dots.f'^*_{p-1}(X_{p-2}).f'^*_p(X_{p-1}).\overline{Y} \quad . \tag{4}$$

This expression is most efficiently computed from right to left, because matrix×vector products are cheaper than matrix×matrix products. This is the *adjoint mode* of AD, most effective for optimization, data assimilation [32], adjoint problems [24], or inverse problems.

Adjoint AD builds a very efficient program [26, Section 3.3], which computes the gradient in a time independent from the number of parameters n. In contrast, computing the same gradient with the *tangent mode* would require running the tangent differentiated program n times.

However, the X_k are required in the *inverse* of their computation order. If the original program *overwrites* a part of X_k, the differentiated program must restore X_k before it is used by $f'^*_{k+1}(X_k)$. Therefore, the central research problem of adjoint AD is to make the X_k available in reverse order at the cheapest cost, using strategies that combine storage, repeated forward computation from available previous values, or even inverted computation from available later values.

Another research issue is to make the AD model cope with the constant evolution of modern language constructs. From the old days of Fortran77, novelties include pointers and dynamic allocation, modularity, structured data types, objects, vectorial notation and parallel programming. We keep developing our models and tools to handle these new constructs.

3.2 Static Analysis and Transformation of programs

Participants: Laurent Hascoët, Jean-Luc Bouchot.

Glossary

abstract syntax tree Tree representation of a computer program, that keeps only the semantically significant information and abstracts away syntactic sugar such as indentation, parentheses, or separators.

control flow graph Representation of a procedure body as a directed graph, whose nodes, known as basic blocks, each contain a sequence of instructions and whose arrows represent all possible control jumps that can occur at run-time.

abstract interpretation Model that describes program static analysis as a special sort of execution, in which all branches of control switches are taken concurrently, and where computed values are replaced by abstract values from a given *semantic domain*. Each particular analysis gives birth to a specific semantic domain.

data flow analysis Program analysis that studies how a given property of variables evolves with execution of the program. Data Flow analysis is static, therefore studying all possible run-time behaviors and making conservative approximations. A typical data-flow analysis is to detect, at any location in the source program, whether a variable is initialized or not.

The most obvious example of a program transformation tool is certainly a compiler. Other examples are program translators, that go from one language or formalism to another, or optimizers, that transform a program to make it run better. AD is just one such transformation. These tools share the technological basis that lets them implement the sophisticated analyses [15] required. In particular there are common mathematical models to specify these analyses and analyze their properties.

An important principle is *abstraction*: the core of a compiler should not bother about syntactic details of the compiled program. The optimization and code generation phases must be independent from the particular input programming language. This is generally achieved using language-specific *front-ends*, language-independent *middle-ends*, and target-specific *back-ends*. In the middle-end, analysis can concentrate on the semantics of a reduced set of constructs. This analysis operates on an abstract representation of programs made of one *call graph*, whose nodes are themselves *flow graphs* whose

nodes (*basic blocks*) contain abstract *syntax trees* for the individual atomic instructions. To each level are attached symbol tables, nested to capture scoping.

Static program analysis can be defined on this internal representation, which is largely language independent. The simplest analyses on trees can be specified with inference rules [18, 27, 16]. But many *data-flow analyses* are more complex, and better defined on graphs than on trees. Since both call graphs and flow graphs may be cyclic, these global analyses will be solved iteratively. *Abstract Interpretation* [19] is a theoretical framework to study complexity and termination of these analyses.

Data flow analyses must be carefully designed to avoid or control combinatorial explosion. At the call graph level, they can run bottom-up or top-down, and they yield more accurate results when they take into account the different call sites of each procedure, which is called *context sensitivity*. At the flow graph level, they can run forwards or backwards, and yield more accurate results when they take into account only the possible execution flows resulting from possible control, which is called *flow sensitivity*.

Even then, data flow analyses are limited, because they are static and thus have very little knowledge of actual run-time values. Far before reaching the very theoretical limit of *undecidability*, one reaches practical limitations to how much information one can infer from programs that use arrays [35, 20] or pointers. Therefore, conservative *over-approximations* must be made, leading to derivative code less efficient than ideal.

3.3 Algorithmic Differentiation and Scientific Computing

Participants: Alain Dervieux, Laurent Hascoët, Bruno Koobus, Stephen Wornom, Jean-Luc Bouchot.

Glossary

linearization In Scientific Computing, the mathematical model often consists of Partial Differential Equations, that are discretized and then solved by a computer program. Linearization of these equations, or alternatively linearization of the computer program, predict the behavior of the model when small perturbations are applied. This is useful when the perturbations are effectively small, as in acoustics, or when one wants the sensitivity of the system with respect to one parameter, as in optimization.

adjoint state Consider a system of Partial Differential Equations that define some characteristics of a system with respect to some parameters. Consider one particular scalar characteristic. Its sensitivity (or gradient) with respect to the parameters can be defined by means of *adjoint* equations, deduced from the original equations through linearization and transposition. The solution of the adjoint equations is known as the adjoint state.

Scientific Computing provides reliable simulations of complex systems. For example it is possible to *simulate* the steady or unsteady 3D air flow around a plane that captures the physical phenomena of shocks and turbulence. Next comes *optimization*, one degree higher in complexity because it repeatedly simulates and applies gradient-based optimization steps until an optimum is reached. The next sophistication is *robustness*, that detects undesirable solutions which, although maybe optimal, are very sensitive to uncertainty on design parameters or on manufacturing tolerances. This makes second derivatives come into play. Similarly *Uncertainty Quantification* can use second derivatives to evaluate how uncertainty on the simulation inputs imply uncertainty on its outputs.

To obtain this gradient and possibly higher derivatives, we advocate adjoint AD (*cf.* 3.1) of the program that discretizes and solves the direct system. This gives the exact gradient of the discrete function computed by the program, which is quicker and more sound than differentiating the original mathematical equations [24]. Theoretical results [23] guarantee convergence of these derivatives when the direct program converges. This approach is highly mechanizable. However, it requires careful study and special developments of the AD model [28, 33] to master possibly heavy memory usage. Among these

additional developments, we promote in particular specialized AD models for Fixed-Point iterations [25, 17], efficient adjoints for linear algebra operators such as solvers, or exploitation of parallel properties of the adjoint code.

4 Application domains

4.1 Algorithmic Differentiation

Algorithmic Differentiation of programs gives sensitivities or gradients, useful for instance for :

- optimum shape design under constraints, multidisciplinary optimization, and more generally any algorithm based on local linearization,
- inverse problems, such as parameter estimation and in particular 4Dvar data assimilation in climate sciences (meteorology, oceanography),
- first-order linearization of complex systems, or higher-order simulations, yielding reduced models for simulation of complex systems around a given state,
- adaptation of parameters for classification tools such as Machine Learning systems, in which Adjoint Differentiation is also known as *backpropagation*.
- mesh adaptation and mesh optimization with gradients or adjoints,
- equation solving with the Newton method,
- sensitivity analysis, propagation of truncation errors.

4.2 Multidisciplinary optimization

A CFD program computes the flow around a shape, starting from a number of inputs that define the shape and other parameters. On this flow one can define optimization criteria e.g. the lift of an aircraft. To optimize a criterion by a gradient descent, one needs the gradient of the criterion with respect to all inputs, and possibly additional gradients when there are constraints. Adjoint AD is the most efficient way to compute these gradients.

4.3 Inverse problems and Data Assimilation

Inverse problems aim at estimating the value of hidden parameters from other measurable values, that depend on the hidden parameters through a system of equations. For example, the hidden parameter might be the shape of the ocean floor, and the measurable values of the altitude and velocities of the surface. Figure 1 shows an example of an inverse problem using the glaciology code ALIF (a pure C version of ISSM [31]) and its AD-adjoint produced by Tapenade.

One particular case of inverse problems is *data assimilation* [32] in weather forecasting or in oceanography. The quality of the initial state of the simulation conditions the quality of the prediction. But this initial state is not well known. Only some measurements at arbitrary places and times are available. A good initial state is found by solving a least squares problem between the measurements and a guessed initial state which itself must verify the equations of meteorology. This boils down to solving an adjoint problem, which can be done though AD [34]. The special case of *4Dvar* data assimilation is particularly challenging. The 4^{th} dimension in "4D" is time, as available measurements are distributed over a given assimilation period. Therefore the least squares mechanism must be applied to a simulation over time that follows the time evolution model. This process gives a much better estimation of the initial state, because both position and time of measurements are taken into account. On the other hand, the adjoint problem involved is more complex, because it must run (backwards) over many time steps. This demanding application of AD justifies our efforts in reducing the runtime and memory costs of AD adjoint codes.

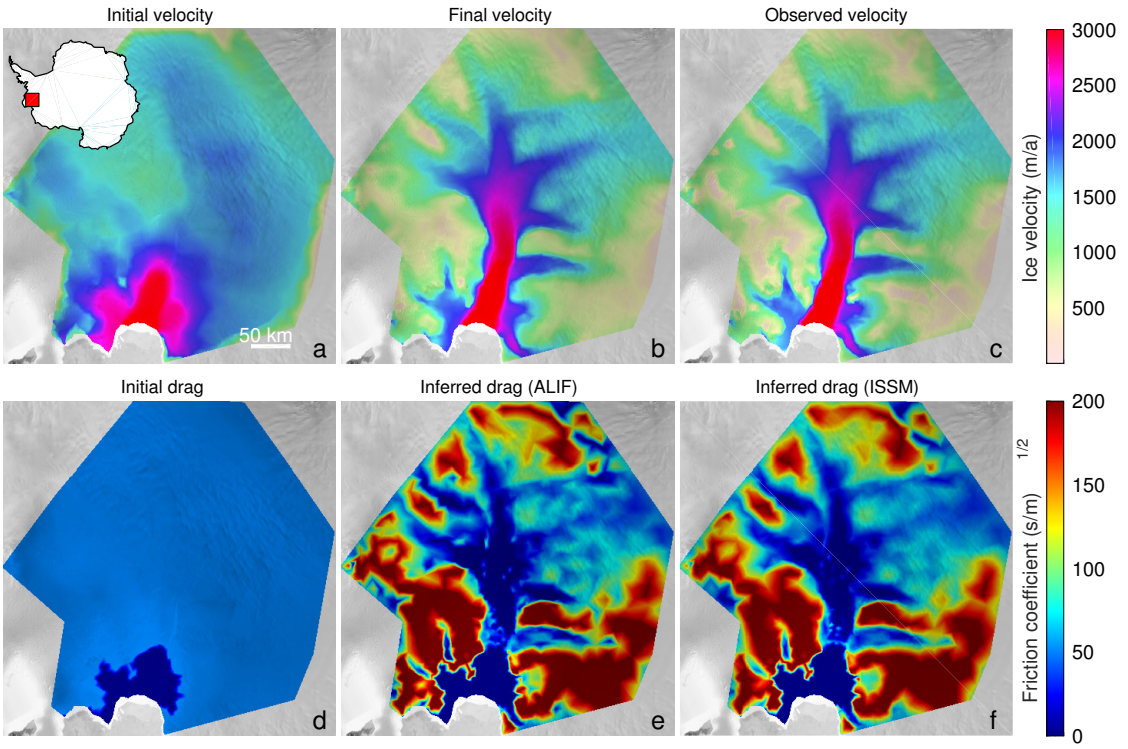

Figure 1: Assimilation of the basal friction under Pine Island glacier, West Antarctica. The final simulated surface velocity (b) is made to match the observed surface velocity (c), by estimation of the basal friction (e). A reference basal friction (f) is obtained by another data assimilation using the hand-written adjoint of ISSM

4.4 Linearization

Simulating a complex system often requires solving a system of Partial Differential Equations. This can be too expensive, in particular for real-time simulations. When one wants to simulate the reaction of this complex system to small perturbations around a fixed set of parameters, there is an efficient approximation: just suppose that the system is linear in a small neighborhood of the current set of parameters. The reaction of the system is thus approximated by a simple product of the variation of the parameters with the Jacobian matrix of the system. This Jacobian matrix can be obtained by AD. This is especially cheap when the Jacobian matrix is sparse. The simulation can be improved further by introducing higher-order derivatives, such as Taylor expansions, which can also be computed through AD. The result is often called a *reduced model*.

4.5 Mesh adaptation

Some approximation errors can be expressed by an adjoint state. Mesh adaptation can benefit from this. The classical optimization step can give an optimization direction not only for the control parameters, but also for the approximation parameters, and in particular the mesh geometry. The ultimate goal is to obtain optimal control parameters up to a precision prescribed in advance.

5 Social and environmental responsibility

5.1 Impact of research results

Our research has an impact on environmental research: in Earth sciences, our gradients are used in inverse problems, to determine key properties in oceanography, glaciology, or climate models. For instance they determine basal friction coefficients of glaciers that are necessary to simulate their future evolution. Another example is to locate sources and sinks of CO2 by coupling atmospheric models and remote measurements.

6 New software, platforms, open data

6.1 New software

6.1.1 AIRONUM

Keywords: Computational Fluid Dynamics, Turbulence

Functional Description: Aironum is an experimental software that solves the unsteady compressible Navier-Stokes equations with k-epsilon, LES-VMS and hybrid turbulence modelling on parallel platforms, using MPI. The mesh model is unstructured tetrahedrization, with possible mesh motion.

URL: https://imag.umontpellier.fr/~koobus/norma.html

Contact: Alain Dervieux

Participant: Alain Dervieux

6.1.2 TAPENADE

Name: Tapenade Automatic Differentiation Engine

Keywords: Static analysis, Optimization, Compilation, Gradients

Scientific Description: Tapenade implements the results of our research about models and static analyses for AD. Tapenade can be downloaded and installed on most architectures. Alternatively, it can be used as a web server. Higher-order derivatives can be obtained through repeated application.

Tapenade performs sophisticated data-flow analysis, flow-sensitive and context-sensitive, on the complete source program to produce an efficient differentiated code. Analyses include Type-Checking, Read-Write analysis, and Pointer analysis. AD-specific analyses include the so-called Activity analysis, Adjoint Liveness analysis, and TBR analysis.

Functional Description: Tapenade is an Algorithmic Differentiation tool that transforms an original program into a new program that computes derivatives of the original program. Algorithmic Differentiation produces analytical derivatives, that are exact up to machine precision. Adjoint-mode AD can compute gradients at a cost which is independent from the number of input variables. Tapenade accepts source programs written in Fortran77, Fortran90, or C. It provides differentiation in the following modes: tangent, vector tangent, adjoint, and vector adjoint.

News of the Year: Major speedup of the static data-flow analysis components. Automatic and asynchronous file offload and prefetching of the adjoint data stack. Improvements of the adjoint differentiation of fixed-point loops. Continued refactoring and bug fixes.

URL: https://team.inria.fr/ecuador/en/tapenade/

Contact: Laurent Hascoët

Participants: Laurent Hascoët, Jean-Luc Bouchot, Jan Hueckelheim, Michael Vossbeck

7 New results

7.1 Profiling for improving Checkpointing schemes

Participants: Laurent Hascoët, Jean-Luc Bouchot, Shreyas Gaikwad *(Univ. of Texas, Austin, USA)*, Sri Hari Krishna Narayanan *(Argonne National Lab, Illinois, USA)*, Jan Hueckelheim *(Argonne National Lab., Illinois, USA)*.

Data-Flow inversion is at the heart of the computation of gradients through AD. This amounts to providing (many of) the intermediate values computed at run-time, in reverse order. The bottleneck is the memory space needed to store these values before retrieving them in reverse order. Alternatively, these values may be recomputed when needed, in which case the bottleneck becomes a quadratic increase of run time. The classical answer is a memory/recomputation trade-off known as checkpointing. By applying checkpointing to a well-chosen collection of nested portions of the code, one can compute its gradient at reasonable memory and run-time costs. However, there is no general approach to find a good (let alone optimal) set of nested checkpoints.

A checkpoint can be placed on any closed sequence of code instructions, such as a procedure or any part of a procedure with a single entry location and a single exit location. The number of possible sets of nested checkpoints is immense and an exhaustive optimal search is therefore out of reach. Yet, we observe that performance of gradient evaluation strongly depends on the set of checkpoints. We observed factors up to 10 or more on large applications.

In order to iteratively improve performance of a gradient computation by changing the choice of checkpoints, we developed a run-time profiling mechanism. Based on a current selection of checkpoints, profiling can estimate the effect of removing each of the current checkpoints, in terms of run-time and memory gain or loss. This estimation is performed during one run of the gradient code, with a minimal overhead.

We implemented this profiling tool inside our AD tool Tapenade, and experimented on two large applications taken from the MIT-GCM test suites. We observed run-time gains up to a factor 4. This work [13] was presented at the 8^{th} International Conference on AD, Chicago USA, September 2024.

7.2 Frontiers of the AD model

Participants: Laurent Hascoët, Jean-Luc Bouchot, Jan Hueckelheim *(Argonne National Lab., Illinois, USA)*.

AD suffers from numerous limitations. Some limitations come from the constraints of application languages, or from coding styles adverse to AD or at least that make it inefficient. A crude distinction, debatable but still quite valid, can be made between AD by execution of a new transformed source code on one hand, and AD by delayed evaluation of an execution trace on the other hand. This distinction is all the more important for the reverse AD mode, that computes gradients.

Creating a new source code that computes the gradient requires a complex tool akin to a compiler, implying a costly development that depends of the targetted application language. However, the resulting new source code can be optimized at compile time, and its resulting performance is only limited by the unavoidable cost of data-flow inversion (see 7.1). On the other hand, an execution trace can be built by a relatively simple instrumentation of the original code, almost independently of the application language, be it heavily templated C++ or one of the newer popular dynamic languages. However, the resulting execution trace (or "tape") is in general huge, as it contains not only the data needed for data-flow inversion, but also the full unrolled sequence of run-time instructions.

This year, we started to explore AD by source-transformation for a dynamic language such as Julia. Our goal is to explore, and hopefully lift, the restrictions that the constructs of Julia impose on source-transformation AD. We are still at a preliminary stage. Difficulties arise in particular from:

- the absence of explicit type declarations.
- the type structure and the special flavor of overloading known as "multiple dispatch".
- the interpreted nature of the language, combined with hidden JIT compilation.
- the presence of macros that dynamically rewrite the source.

No differentiated code was produced yet, but a few test codes could be parsed and analyzed.

Other limitations of AD can be of a quite abstract nature, and may come for instance from the mathematical equations underlying the code, or from the fundamental difference between computer's float numbers and real numbers. Categorizing these limitations is difficult, and a (very long) catalogue may prove useless. Jan Hueckelheim considered the question of what one can reasonably expect from AD. The assumption is that problematic cases are often not about AD producing wrong derivatives, but rather meaningful derivatives, only for a model different from what the user has in mind. Accurate knowledge of the context in which AD derivatives are meaningful will help users build a realistic expectation for AD tools. A joint article "A taxonomy of automatic differentiation pitfalls" [11] was published this year.

7.3 Application to large industrial codes

Participants: Laurent Hascoët, Sébastien Bourasseau *(ONERA)*, Cedric Content *(ONERA)*, Bruno Maugars *(ONERA)*, Shreyas Gaikwad *(Univ. of Texas, Austin, USA)*, Sri Hari Krishna Narayanan *(Argonne National Lab., Illinois, USA)*, Michael Vossbeck *(The Inversion Lab, Hamburg, Germany)*.

We support users of Algorithmic Differentiation of large codes.

On CFD, we continue support for the use of Tapenade to provide gradient computations in the Onera Platform "SoNice". SoNice is a new CFD platform developed at ONERA by Bourasseau, Maugars, Content and colleagues, as a successor to Elsa. In the modular architecture of SoNice, Tapenade is used to build the differentiated counterpart of each module or operator. The implementation of each module is devised to be platform-independent, eventually targetting several multithreaded architectures through e.g. OpenMP or Cuda. This imposes technical constraints on the language differentiated by Tapenade, at various abstraction levels that range from AD models for multithreaded code, down to ways of handling

`fpp` and `cpp` directives and automatically setting entry points and labels in a differentiated code to allow for some necessary post-processing.

On Earth science, Shreyas Gaikwad continued building gradient computations on more test cases of the MIT-GCM, using Tapenade, with support from Krishna Narayanan from Argonne. These applications range from atmospheric to oceanic science and glaciology. This work is now published in [22]. Also on Earth science, we support adjoint differentiation of the BEPS [29] code, a "biosphere" simulation model taking vegetation into account. This code is developed in part by the small company The Inversion Lab in Hamburg, Germany.

In Particle Physics, Krishna Narayanan differentiated with Tapenade the code HFBTHO from Lawrence Livermore National Lab., in tangent AD mode, which is a nuclear energy density functional (EDF) solver used to model the structure of atomic nuclei. This is a large Fortran code that unveiled a few technical issues but no major concern for Tapenade. An article is in preparation. We also successfully differentiated one test configuration of QCDNUM, a simulation code for Quantum Chromodynamics (QCD), in both tangent and reverse AD modes. This experiment, suggested by Lukas Heinrich and team at TU Munich and Max-Planck Institute, is supposed to lead to further collaboration.

7.4 Aeroacoustics/projet Norma

Participants: Alain Dervieux, Bastien Sauvage, Bruno Koobus *(IMAG, Univ. of Montpellier)*, Stephen Wornom *(IMAG, Univ. of Montpellier)*.

The progress in highly accurate schemes for compressible flows on unstructured meshes (together with advances in massive parallelization of these schemes) allows to solve problems previously out of reach, such as aeroacoustics around rotating machines. The four-years programme Norma, associating

- IMAG of Montpellier University (B. Koobus, coordinator),
- Ecuador of INRIA Sophia-Antipolis,

and supported by the ANR, was active till September 2024. Norma is a cooperation on the subject of extending Computational AeroAcoustics methods to simulate the noise emited by rotating machines (helicopters, aerial vehicles, unmanned aerial vehicles, wind turbines...). A final synthesis of the work is available as [30].

Since flows around rotating machines are turbulent and often characterized by laminar-turbulent transitions. The turbulence model must be able to take these phenomena into account while remaining reasonably expensive and of low dissipation in order to propagate acoustic waves accurately. The new turbulence model developed is an intermittency-based hybrid RANS/LES model (Reynolds averaged Navier-Stokes, Large Eddy Simulation) that addresses the targeted objectives. Flow simulation is based on the combination of an anisotropic mesh adaptation method with a Chimera or a multiple reference frame (MRF) method. On the other hand, and in order to improve the quality and efficiency of the prediction of unsteady flows, a new adaptation method based on a space-time metric is developed. Furthermore, a new adaptation technique, which introduces an adaptation criterion controlling in particular the LES error model, is also proposed for calculations that use hybrid turbulence models which are those favored in this program. Finally, a finite volume type numerical approximation scheme with quadratic or cubic reconstruction of the solution is developed in order to obtain high order accuracy on an unstructured mesh.

A detailed description with progress reports is available at this location.

7.5 Rotating machines

Participants: Alain Dervieux, Didier Chargy *(Lemma, Sophia-Antipolis)*, Bastien Sauvage, Bruno Koobus *(IMAG, Univ. of Montpellier)*.

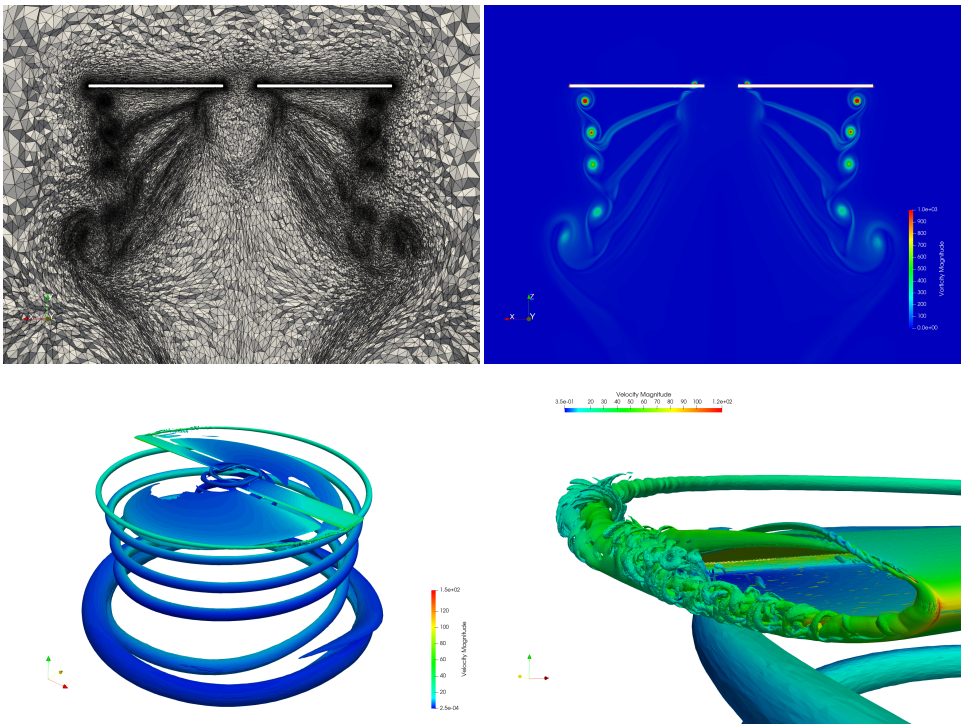

Figure 2: Example simulation of rotating machines : hybrid/DDES computation around the Caradonna-Tung helicopter rotor: adapted mesh, vorticity field, iso-surface of the Q-factor, and one detail of it.

An important output of the Norma project is the research of an efficient combination of multiple reference frame (MRF) rotating formulation with anisotropic mesh adaptation. A first version of this combination has been defined and validated by Bastien Sauvage. With this new method, Bastien Sauvage has finalised two unsteady mesh-adaptive computations, one of a single Caradonna-Tung helicoter rotor and one of the combination of the Caradonna-Tung rotor with the Robin fuselage (see Fig. 2). The computation of the single Caradonna-Tung rotor shows a well-formed vortex shedding at the end of the two rotor wings, corresponding to high-frequency sound emission.

7.6 Turbulence models

Participants: Bastien Sauvage, Alain Dervieux, Bruno Koobus *(IMAG, Univ. of Montpellier)*, Stephen Wornom *(IMAG, Univ. of Montpellier)*.

The prediction of academic and industrial turbulent flows progresses with each decade. For example in [21], most results were 2D, RANS computations were very far from mesh convergence, and far from experimental measurements. Also, they did not apply well to many unsteady flows. Since then, new VLES (Very Large Eddy Simulation) and hybrid models appeared for addressing a large class of unsteady flows. At the beginning of 2000's, a first generation of VLES/hybrid methods, including the Detached Eddy Simulation, showed interesting results. It is interesting to measure how these new physical models, and new numerical methods, improved one or two decades later. In association with IMAG, a wide series of test cases was recomputed in order to measure on these cases the progress in turbulence computation obtained thanks to the three novelties introduced by the Montpellier-Sophia cooperation:

- a novel hybrid LES-RANS model with intermittency,
- an fourth-order CENO4 advective approximation for RANS and hybrid calculations,
- the new unsteady mesh adaptation for RANS and hybrid calculations.

An article in preparation ("Computing the flow past a cylinder : influence of models and numerics" by B. Sauvage, F. Miralles, S. Wornom, B. Koobus, A. Dervieux) discusses and contrasts these simulations.

7.7 Space-time mesh adaptation

Participants: Bastien Sauvage, Alain Dervieux, Frédéric Alauzet *(GammaO, INRIA, Saclay)*.

The INRIA teams GammaO (Saclay) and Ecuador (Sophia-Antipolis) have continued to study the application of anisotropic mesh adaptation to unsteady flow simulation. The baseline approach is the Global Transient Fixed Point algorithm combined with a timestep relying on Courant-type stability condition for explicit time-advancing. This remains the best approach for many transient flows. For many unsteady turbulent flows at high Reynolds number, this is however inefficient since very small cells in the boundary layer impose a very small explicit time step. Implicit timestepping is compulsory. Indeed, the time steps which can be used are notably larger than those permitted with an explicit time advancing. Large time steps bring a higher CPU efficiency, but the time approximation accuracy becomes an issue. Too large time steps degrade the prediction, too small time steps increases the computational cost. We developed a novel space and time adaptation method for the implicit time advancing used in turbulence computations. The new method has been extended to a new Global Transient Fixed Point mesh adaptation algorithm addressing turbulent flows with several meshes applied to the time interval. This work has been published in [12].

7.8 Mesh adaptation for LES

Participants: Bastien Sauvage, Alain Dervieux, Bruno Koobus *(IMAG, Univ. of Montpellier)*, Frédéric Alauzet *(GammaO, INRIA, Saclay)*.

With statistical turbulence models, only a part of turbulent industrial flows can be predicted at an affordable cost. The rest may involve large detached flow regions, which cannot be accurately described by statistical modeling, or vortices producing noise, that the engineer wants to predict accurately. The Large Eddy Simulation (LES) is a model which predicts a part of the vortices of industrial interest. LES relies on filtering too small vortices and on modeling their effect on the larger ones. But this approach is one or two orders of magnitude more computer intensive than statistical modeling and therefore cannot be routinely used by engineers. Germano proposed an analysis using two different filters in order to measure the efficicency of a family of LES models. Recently, Toosi and Larsson demonstrated that the Germano analysis in fact deals with the source term of LES modeling error. We propose a novel adaptation method based on the Germano dynamic-LES analysis, taking into account the LES model error. This method was validated on a set of test cases and exhibits a better efficiency. An article is being prepared ("A metric-based mesh adaptation for hybrid RANS/LES flow calculations" by B. Sauvage, B. Koobus, F. Alauzet, A. Dervieux).

8 Partnerships and cooperations

Participants: Laurent Hascoët, Ines Naumann, Uwe Naumann.

8.1 International research visitors

8.1.1 Visits of international scientists

Uwe Naumann

Status researcher

Institution of origin: RWTH Aachen

Country: Germany

Dates: May-June 2024

Context of the visit: Collaboration on AD tools

Mobility program/type of mobility: sabbatical

Ines Naumann

Status Engineer

Institution of origin: KEMPEN KRAUSE INGENIEURE GmbH

Country: Germany

Dates: May-June 2024

Context of the visit: AD Experiments on civil engineering CAD codes

Mobility program/type of mobility: research stay

9 Dissemination

Participants: Laurent Hascoët, Alain Dervieux.

9.1 Promoting scientific activities

9.1.1 Scientific events: organisation

Laurent Hascoët was on the Organization Committee of the 8th International Conference on Automatic Differentiation, held on September 16-19. He was also a member of the Program Committee and one of the reviewers.

9.1.2 Invited talks

Laurent Hascoët gave two invited talks this year,

- during the winter seminar of the CASTOR team, February 18th-21st
- at the ECCO annual meeting, Univ. of Texas at Austin, March 22nd.

9.1.3 Scientific expertise

Jean-Luc Bouchot supports users of our AD tools at ONERA and Dassault Aviation, and visited ONERA in Chatillon on March 28th.

9.2 Teaching - Supervision - Juries

9.2.1 Juries

- Alain Dervieux was supervisor and member of the Jury for the PhD defense of Bastien Sauvage, “Numerical approximation and adaptation for flows in rotating machines”, September 27th.
- Laurent Hascoët was reviewer and member of the Jury for the PhD defense of Max Aehle at TU Kaiserslautern, Germany, “Automatic Differentiation of Compiled Programs”, October 8th.

10 Scientific production

10.1 Major publications

[1] F. Courty, A. Dervieux, B. Koobus and L. Hascoët. 'Reverse automatic differentiation for optimum design: from adjoint state assembly to gradient computation'. In: *Optimization Methods and Software* 18.5 (2003), pp. 615–627.

[2] B. Dauvergne and L. Hascoët. 'The Data-Flow Equations of Checkpointing in reverse Automatic Differentiation'. In: *International Conference on Computational Science, ICCS 2006, Reading, UK*. 2006.

[3] D. Goldberg, S. H. K. Narayanan, L. Hascoët and J. Utke. 'An optimized treatment for algorithmic differentiation of an important glaciological fixed-point problem'. In: *Geoscientific Model Development* 9.5 (2016), p. 27. URL: https://hal.inria.fr/hal-01413295.

[4] L. Hascoët. 'Adjoints by Automatic Differentiation'. In: *Advanced data assimilation for geosciences*. Oxford University Press, 2014. URL: https://hal.inria.fr/hal-01109881 (cit. on p. 3).

[5] L. Hascoët, U. Naumann and V. Pascual. '"To Be Recorded" Analysis in Reverse-Mode Automatic Differentiation'. In: *Future Generation Computer Systems* 21.8 (2004).

[6] L. Hascoët and V. Pascual. 'The Tapenade Automatic Differentiation tool: Principles, Model, and Specification'. In: *ACM Transactions On Mathematical Software* 39.3 (2013). URL: http://dx.doi.org/10.1145/2450153.2450158.

[7] L. Hascoët, J. Utke and U. Naumann. 'Cheaper Adjoints by Reversing Address Computations'. In: *Scientific Programming* 16.1 (2008), pp. 81–92.

[8] L. Hascoët, M. Vázquez, B. Koobus and A. Dervieux. 'A Framework for Adjoint-based Shape Design and Error Control'. In: *Computational Fluid Dynamics Journal* 16.4 (2008), pp. 454–464.

[9] L. Hascoët and J. Utke. 'Programming language features, usage patterns, and the efficiency of generated adjoint code'. In: *Optimization Methods and Software* 31 (2016), pp. 885–903. DOI: 10.1080/10556788.2016.1146269. URL: https://hal.inria.fr/hal-01413332.

[10] J. C. Hueckelheim, L. Hascoët and J.-D. Müller. 'Algorithmic differentiation of code with multiple context-specific activities'. In: *ACM Transactions on Mathematical Software* (2016). URL: https://hal.inria.fr/hal-01413321.

10.2 Publications of the year

International journals

[11] J. Hückelheim, H. Menon, W. Moses, B. Christianson, P. Hovland and L. Hascoët. 'A taxonomy of automatic differentiation pitfalls'. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (2nd Sept. 2024). DOI: 10.1002/widm.1555. URL: https://inria.hal.science/hal-04693411 (cit. on p. 10).

[12] B. Sauvage, F. Alauzet and A. Dervieux. 'A space and time fixed point mesh adaptation method'. In: *Journal of Computational Physics* 519 (Dec. 2024), p. 113389. DOI: 10.1016/j.jcp.2024.113389. URL: https://inria.hal.science/hal-04715086 (cit. on p. 13).

International peer-reviewed conferences

[13] L. Hascoët, J.-L. Bouchot, S. S. Gaikwad, S. H. K. Narayanan and J. Hückelheim. 'Profiling checkpointing schedules in adjoint ST-AD'. In: AD 2024 - 8th International Conference on Algorithmic Differentiation. Chicago, United States, 16th Sept. 2024. URL: https://inria.hal.science/hal-04693450 (cit. on p. 9).

Doctoral dissertations and habilitation theses

[14] B. Sauvage. 'Numerical approximation and adaptation for flows in rotating machines'. Université Côte d'Azur, 27th Sept. 2024. URL: https://theses.hal.science/tel-04810649.

10.3 Cited publications

[15] A. Aho, R. Sethi and J. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986 (cit. on p. 4).

[16] I. Attali, V. Pascual and C. Roudet. *A language and an integrated environment for program transformations*. research report 3313. INRIA, 1997. URL: http://hal.inria.fr/inria-00073376 (cit. on p. 5).

[17] B. Christianson. 'Reverse accumulation and implicit functions'. In: *Optimization Methods and Software* 9.4 (1998), pp. 307–322 (cit. on p. 6).

[18] D. Clément, J. Despeyroux, L. Hascoët and G. Kahn. 'Natural semantics on the computer'. In: *Proceedings, France-Japan AI and CS Symposium, ICOT* (1986). Ed. by K. Fuchi and M. Nivat. Also, Information Processing Society of Japan, Technical Memorandum PL-86-6. Also INRIA research report # 416, pp. 49–89. URL: http://hal.inria.fr/inria-00076140 (cit. on p. 5).

[19] P. Cousot. 'Abstract Interpretation'. In: *ACM Computing Surveys* 28.1 (1996), pp. 324–328 (cit. on p. 5).

[20] B. Creusillet and F. Irigoin. 'Interprocedural Array Region Analyses'. In: *International Journal of Parallel Programming* 24.6 (1996), pp. 513–546 (cit. on p. 5).

[21] A. Dervieux, M. Braza and J.-P. Dussauge. *Computation and comparison of efficient turbulence models for Aeronautics - European Research Project ETMA*. Vol. 65. Vieweg, Braunschweig; Wiesbaden, 1998 (cit. on p. 12).

[22] S. Gaikwad, S.-H.-K. Narayanan, L. Hascoët, J.-M. Campin, H. Pillar, A. Nguyen, J. Hüeckelheim, P. Hovland and P. Heimbach. 'MITgcm-AD v2: Open source tangent linear and adjoint modeling framework for the oceans and atmosphere enabled by the Automatic Differentiation tool Tapenade'. In: *Future Generation Computer Systems* 163 (2015) (cit. on p. 11).

[23] J. Gilbert. 'Automatic differentiation and iterative processes'. In: *Optimization Methods and Software* 1 (1992), pp. 13–21 (cit. on p. 5).

[24] M.-B. Giles. 'Adjoint methods for aeronautical design'. In: *Proceedings of the ECCOMAS CFD Conference*. Swansea, U.K., 2001 (cit. on pp. 4, 5).

[25] A. Griewank and C. Faure. 'Reduced Gradients and Hessians from Fixed Point Iteration for State Equations'. In: *Numerical Algorithms* 30(2) (2002), pp. 113–139 (cit. on p. 6).

[26] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd. SIAM, Other Titles in Applied Mathematics, 2008 (cit. on p. 4).

[27] L. Hascoët. 'Transformations automatiques de spécifications sémantiques: application: Un vérificateur de types incremental'. PhD thesis. Université de Nice Sophia-Antipolis, 1987 (cit. on p. 5).

[28] P. Hovland, B. Mohammadi and C. Bischof. *Automatic Differentiation of Navier-Stokes computations*. Tech. rep. MCS-P687-0997. Argonne National Laboratory, 1997 (cit. on p. 5).

[29] W. Ju, J. Chen, T. Black, A. Barr, J. Liu and B. Chen. 'Modelling multi-year coupled carbon and water fluxes in a boreal aspen forest'. In: *Agric. For. Meteorol.* 140.1-4 (2006), pp. 136–151. DOI: 10.1016/j.agrformet.2006.08.008 (cit. on p. 11).

[30] B. Koobus. *Compte-rendu de fin de projet, Projet NORMA*. Tech. rep. ANR-19-CE40-0020-01, Programme AAPG 2019. 2024 (cit. on p. 11).

[31] E. Larour, J. Utke, B. Csatho, A. Schenk, H. Seroussi, M. Morlighem, E. Rignot, N. Schlegel and A. Khazendar. 'Inferred basal friction and surface mass balance of the Northeast Greenland Ice Stream using data assimilation of ICESat (Ice Cloud and land Elevation Satellite) surface altimetry and ISSM (Ice Sheet System Model)'. In: *Cryosphere* 8.6 (2014), pp. 2335–2351. DOI: 10.5194/tc-8-2335-2014. URL: http://www.the-cryosphere.net/8/2335/2014/ (cit. on p. 6).

[32] F.-X. Le Dimet and O. Talagrand. 'Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects'. In: *Tellus* 38A (1986), pp. 97–110 (cit. on pp. 4, 6).

[33] B. Mohammadi. 'Practical application to fluid flows of automatic differentiation for design problems'. In: *Von Karman Lecture Series* (1997) (cit. on p. 5).

[34] N. Rostaing. ‘Différentiation Automatique: application à un problème d’optimisation en météorologie’. PhD thesis. université de Nice Sophia-Antipolis, 1993 (cit. on p. 6).

[35] R. Rugina and M. Rinard. ‘Symbolic Bounds Analysis of Pointers, Array Indices, and Accessed Memory Regions’. In: *Proceedings of the ACM SIGPLAN’00 Conference on Programming Language Design and Implementation*. ACM, 2000 (cit. on p. 5).