Activity Report 2018

Project-Team ECUADOR

Program transformations for scientific computing

RESEARCH CENTER
Sophia Antipolis - Méditerranée

THEME
Numerical schemes and simulations

Table of contents

Project-Team ECUADOR

Creation of the Project-Team: 2014 January 01

Keywords:

Computer Science and Digital Science:

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, Visitors, External Collaborators

Research Scientists

Laurent Hascoët [Team leader, Inria, Senior Researcher, HDR]
Alain Dervieux [Inria, Emeritus, HDR]
Valérie Pascual [Inria, Researcher]

PhD Student

Eléonore Gauci [Education Nationale]

Administrative Assistant

Christine Claux [Inria]

Visiting Scientist

Olivier Allain [LEMMA]

External Collaborators

Emmanuelle Itam [CNAM]
Bruno Koobus [Univ Montpellier II (sciences et techniques du Languedoc)]
Stephen Wornom [LEMMA]

2. Overall Objectives

2.1. 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 want 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 now.

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 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, Valérie Pascual.

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 [2].

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

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

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

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

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:

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

  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:

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

  This expression is most efficiently computed from right to left, because matrix$\times$vector products are cheaper than matrix$\times$matrix products. This is the *adjoint mode* of AD, most effective for optimization, data assimilation [28], adjoint problems [21], or inverse problems.

Adjoint AD builds a very efficient program [24], 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, Valérie Pascual.

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 [14] 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 [17], [25], [15]. 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* [18] 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 [31], [19] 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, Eléonore Gauci, Emmanuelle Itam, Olivier Allain, Stephen Wornom.

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 derivative come into play. Similarly *Uncertainty Quantification* can use second derivatives to evaluate how uncertainty on the simulation inputs imply uncertainty on its outputs.

We investigate several approaches to obtain the gradient, between two extremes:

- One can write an *adjoint system* of mathematical equations, then discretize it and program it by hand. This is time consuming. Although this looks mathematically sound [21], this does not provide the gradient of the discretized function itself, thus degrading the final convergence of gradient-descent optimization.
- One can apply adjoint AD (*cf* 3.1) on the program that discretizes and solves the direct system. This gives exactly the adjoint of the discrete function computed by the program. Theoretical results [20] guarantee convergence of these derivatives when the direct program converges. This approach is highly mechanizable, but leads to massive use of storage and may require code transformation by hand [26], [29] to reduce memory usage.

If for instance the model is steady, or when the computation uses a Fixed-Point iteration, tradeoffs exist between these two extremes [22], [16] that combine low storage consumption with possible automated adjoint generation. We advocate incorporating them into the AD model and into the AD tools.

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,
- adaption 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 [27]) and its AD-adjoint produced by Tapenade.

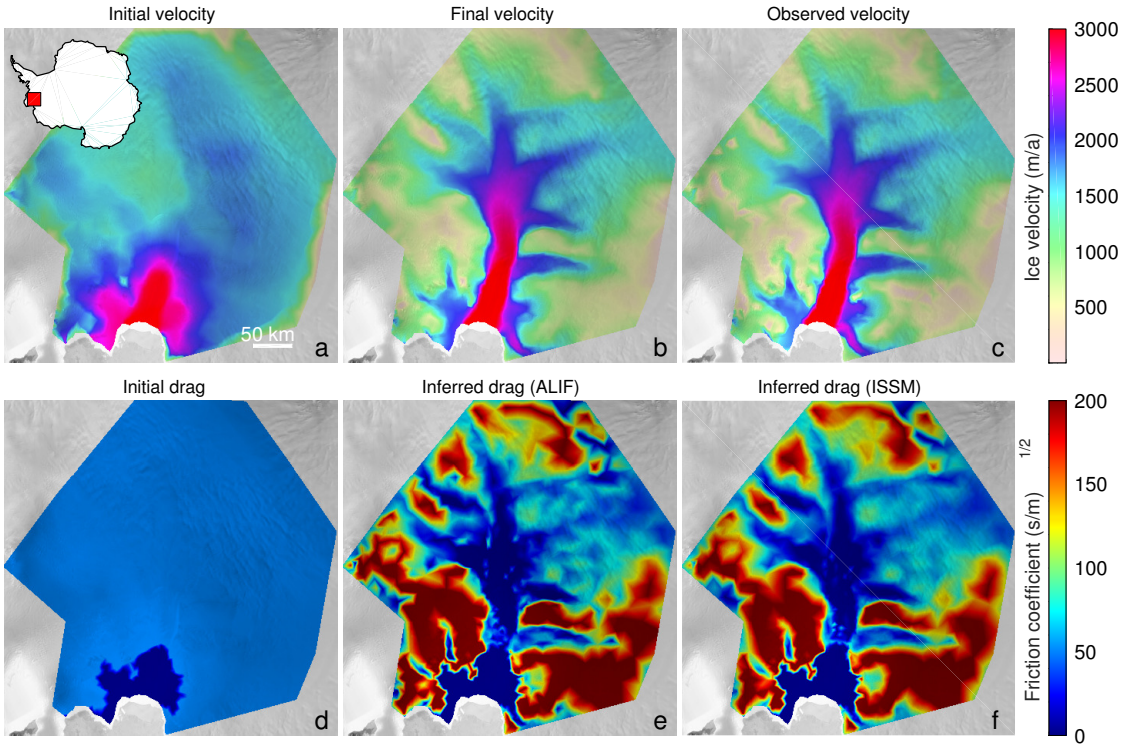

Figure 1. Assimilation of the basal friction under Pine Island glacier, West Antarctica. The final simulated surface velocity (b) is made to match the observed surface velocity (c), by estimation of the basal friction (e). A reference basal friction (f) is obtained by another data assimilation using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

One particular case of inverse problems is *data assimilation* [28] 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 [30]. 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.

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. New Software and Platforms

5.1. 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.

- Participant: Alain Dervieux
- Contact: Alain Dervieux
- URL: http://www-sop.inria.fr/tropics/aironum

5.2. TAPENADE

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:

Activity analysis: Detects variables whose derivative is either null or useless, to reduce the number of derivative instructions.

Adjoint Liveness analysis: Detects the source statements that are dead code for the computation of derivatives.

TBR analysis: In adjoint-mode AD, reduces the set of source variables that need to be recovered.
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: - Continued development of multi-language capacity: AD of codes mixing Fortran and C - Continued front-end for C++ based on Clang - Experimental support for building Abs-Normal Form tangent of non-smooth codes

- Participants: Laurent Hascoët and Valérie Pascual
- Contact: Laurent Hascoët
- URL: http://www-sop.inria.fr/tropics/tapenade.html

6. New Results

6.1. Towards Algorithmic Differentiation of C++

Participants: Laurent Hascoët, Valérie Pascual, Frederic Cazals [ABS team, Inria Sophia-Antipolis].

We made progress towards the extension of Tapenade for C++. Last year, an external parser for C++ was built on top of Clang-LLVM https://clang.llvm.org/ and connected to the input formalism "IL" of Tapenade, but the internals of Tapenade were not able to handle the new constructs present in this input. This year, integration of C++ was pushed further by taking into account many of the new constructs (namespaces, classes, constructors and destructors) in the Internal Representation(IR) of Tapenade. Not surprisingly, this implied deep changes in several areas of Tapenade code. The IR of Tapenade now contains classes, constructors and destructors, and also has a faithful representation for namespaces. The textual nested structure and the control-flow parts of the IR are correct. The symbol tables and the representation for memory locations are still under development.

As a result, Tapenade is now able to input its first C++ files and is able to output them, but without transformation. Although not advertised nor documented, the functionality is present in the latest release 3.14. Data-Flow analysis and code transformation (e.g. AD) will not be possible until we have a correct IR about variables and their memory locations. This work is going on.

This work benefited from the expertise in C++ of Frederic Cazals (Inria ABS team). The ABS team provided a large test application code (SBL, https://sbl.inria.fr/) for Molecular Dynamics, which will be our first C++ target.

6.2. AD of mixed-language codes

Participants: Valérie Pascual, Laurent Hascoët.

Last year Tapenade was extended to differentiate codes that mix different languages, beginning with the tangent mode of AD. Our motivating application here is Calculix, a 3-D Structural Finite Element code that mixes Fortran and C. This year, we continued development towards Adjoint Differentiation. Although more complete testing is needed, we now have a first correct adjoint of Calculix.

Tapenade can now routinely differentiate Fortran+C codes, and accepts and takes advantage of the interoperability directives provided by the Fortran 2003 standard. It can handle not only procedure parameters correspondence, but also interoperability between C `struct` and Fortran `COMMON` blocks. Laurent Hascoët presented the advancement of this work at the ISMP 2018 congress in Bordeaux https://ismp2018.sciencesconf.org/.

C files (aka "translation units") and Fortran modules are two instances of the more general notion of "package" for which we are looking for a unified representation in Tapenade. It appears that this common representation could also handle C++ namespaces.

6.3. Differentiation of non-smooth programs

Participants: Laurent Hascoët, Sri Hari Krishna Narayanan [Argonne National Lab. (Illinois, USA)].

Algorithmic Differentiation can be used to derive tangent models that cope with a certain class of non-smoothness, through the use of the so-called Abs-Normal Form (ANF) [23]. These tangent models incorporate some knowledge of the nearby discontinuities of the derivatives. These models bring some additional power to processes that use tangent approximations, such as simulation, optimization, or solution of differential equations.

The mechanics to derive these special tangent models can be built as an extension of standard tangent linear Algorithmic Differentiation. This has been first demonstrated by the AD tool AdolC which, being based on Operator Overloading, is more flexible and seems a natural choice for implementation. Together with Krishna Narayanan, we recently tried a similar adaption on Source-Transformation AD tools. It appears that very little development is needed in the AD-tool. Specifically for Tapenade, it appears that no development at all is needed in the tool itself. Any end-user can already produce ANF tangent without needing any access to the tool source. All it requires is a customized derivative of the absolute-value function (ABS), which is currently less than 40 lines of code.

Building the ANF of a given program introduces one new variable per run-time execution of the ABS function. As the number of rows and columns of the constructed extended Jacobian both grow like the number of variables, it may become unreasonably large for large codes. To overcome this issue, we explore the possibility of finding at run-time the "important" ABS calls that deserve this treatment, and those that don't. We base this decision on a notion of distance to the kink induced by this ABS call as illustrated by Figure 2. We presented these experiments at a Shonan meeting on this question (Shonan, Japan, June 25-29) and at a workshop of ISMP 2018 (Bordeaux, July 2-6)

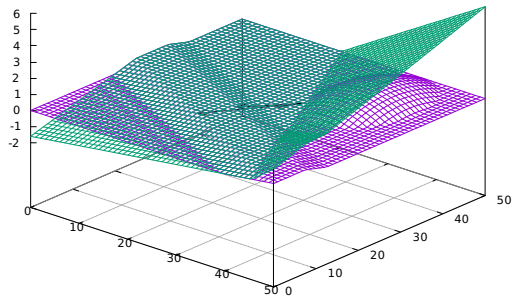

Figure 2. Abs-Normal Form of a non-smooth function: purple:original function, green ANF computed at (30,30). The ANF is linearized around the point of interest, and at the same time captures the non-smooth behavior. Notice the ANF divergence from the original function on the left, due to neglecting the leftmost kink which was decided "far" enough from the point of interest. The ANF divergence on the right is the natural effect of linearization

6.4. AD-adjoints and C dynamic memory management

Participants: Laurent Hascoët, Sri Hari Krishna Narayanan [Argonne National Lab. (Illinois, USA)].

One of the current frontiers of AD research is the definition of an adjoint AD model that can cope with dynamic memory management. This research is central to provide reliable adjoint differentiation of C, and for our distant goal of AD of C++. This research is conducted in collaboration with the MCS department of Argonne National Lab. Our partnership is formalized by joint participation in the Inria joint lab JLESC, and partly funded by the Partner University Fund (PUF) of the French embassy in the USA.

Adjoint AD must reproduce in reverse order the control decisions of the original code. In languages such as C, allocation of dynamic memory and pointer management form a significant part of these control decisions. Reproducing memory allocation in reverse means reallocating memory, possibly receiving a different memory chunk. Reproducing pointer addresses in reverse thus requires to convert addresses in the former memory chunks into equivalent addresses in the new reallocated chunks. Together with Krishna Narayanan from Argonne, we experiment on real applications to find the most efficient solution to this address conversion problem. We jointly develop a library (called ADMM, ADjoint Memory Management) whose primitives are used in AD adjoint code to handle this address conversion. Both our AD tool Tapenade and Argonne's tool OpenAD use ADMM in the adjoint code they produce.

This year, trying to prove correctness of our current address conversion, we discovered some limitations that indeed made the proof impossible. To solve these issues, it seems necessary to assign at run-time a unique identifier to each chunk of memory used by the code, and to carry this identifier along with every pointer. This results in a code transformation which, although more complex than expected, can still be described by a small set of rewrite rules. Moreover, this alternative method should reduce the run-time overhead that we observed previously. Implementation and measurements are still under way. We presented this recent research in the form of a catalogue of alternatives for Data-Flow reversal of memory addresses, at the 21st EuroAD workshop (Jena, Germany, November 19-20).

6.5. Application to large industrial codes

Participants: Valérie Pascual, Laurent Hascoët, Bruno Maugars [ONERA], Sébastien Bourasseau [ONERA], Bérenger Berthoul [ONERA].

We support industrial users with their first experiments of Algorithmic Differentiation of large in-house codes.

This year's main application is with ONERA on their ElsA CFD platform (Fortran 90). Both tangent and adjoint models of the kernel of ElsA were built successfully by Tapenade. It is worth noticing that this application was performed inside ONERA by ONERA engineers (Bruno Maugars, Sébastien Bourasseau, Bérenger Berthoul) with no need for installation of ElsA inside Inria. We take this as a sign of maturity of Tapenade. Apart from a few minor corrections, our contributon was essentially during development meetings, to point out some strategies and tool options to obtain efficient differentiated code. One emphasis was on adjoint of vectorized code, which was produced as vectorized code too by means of a seldom-used Tapenade option that stores intermediate values statically, i.e. not on a global stack. Sébastien Bourasseau presented the first results at the 21st EuroAD workshop (Jena, Germany, November 19-20), with convincing performance on industrial-size test cases. A joint article is in preparation.

6.6. Multirate methods

Participants: Alain Dervieux, Bruno Koobus, Emmanuelle Itam, Stephen Wornom.

This study is performed in collaboration with IMAG-Montpellier. It addresses an important complexity issue in unsteady mesh adaptation and took place in the work done in the ANR Maidesc (ended 2017). Unsteady high-Reynolds computations are strongly penalized by the very small time step imposed by accuracy requirements on regions involving small space-time scales. Unfortunately, this is also true for sophisticated unsteady mesh adaptive calculations. This small time step is an important computational penalty for mesh adaptive methods of AMR type. This is also the case for the Unsteady Fixed-Point mesh adaptive methods developed by Ecuador in cooperation with the Gamma3 team of Inria-Saclay. In the latter method, the loss of efficiency is even more crucial when the anisotropic mesh is locally strongly stretched since only very few cells are in the regions of small time-step constraint. This loss is evaluated as limiting the numerical convergence order for

discontinuities to 8/5 instead of second-order convergence. An obvious remedy is to design time-consistent methods using different time steps on different parts of the mesh, as far as they are efficient and not too complex. The family of time-advancing methods in which unsteady phenomena are computed with different time steps in different regions is referred to as the multirate methods. In our collaboration with university of Montpellier, a novel multirate method using cell agglomeration has been designed and developed in our AIRONUM CFD platform. A series of large-scale test cases show that the new method is much more efficient than an explicit method, while retaining a similar time accuracy over the whole computational domain. A novel analysis shows that the proposed multirate algorithm indeed solves the unsteady mesh adaptation barrier identified in previous works. This work is being published in a journal [13].

6.7. Control of approximation errors

Participants: Eléonore Gauci, Alain Dervieux, Adrien Loseille [Gamma3 team, Inria-Rocquencourt], Frédéric Alauzet [Gamma3 team, Inria-Rocquencourt], Anca Belme [university of Paris 6], Gautier Brèthes [university of Montreal], Alexandre Carabias [Lemma].

Reducing approximation errors as much as possible is a particular kind of optimal control problem. We formulate it exactly this way when we look for the optimal metric of the mesh, which minimizes a user-specified functional (goal-oriented mesh adaptation). In that case, the usual methods of optimal control apply, using adjoint states that can be produced by Algorithmic Differentiation.

This year, two conference papers were written on the methods of the team, including new analyses in [11],[10], a work on correctors in CFD in an AIAA paper. A detailed study of adjoint-based mesh adaptation for Navier-Stokes flows has been completed and published in a journal [9].

Following participation of Gamma3 and Ecuador to the European project UMRIDA (ended 2017), we wrote chapters 20, 21, 45, and 48 of the book "Uncertainty Management for Robust Industrial Design in Aeronautics", edited by C. Hirsch et al. in the Springer series Notes on Numerical Fluid Mechanics and Multidisciplinary Design (2019).

6.8. Turbulence models

Participants: Alain Dervieux, Bruno Koobus, Stephen Wornom, Maria-Vittoria Salvetti [University of Pisa].

Modeling turbulence is an essential aspect of CFD. The purpose of our work in hybrid RANS/LES (Reynolds Averaged Navier-Stokes / Large Eddy Simulation) is to develop new approaches for industrial applications of LES-based analyses. In the applications targetted (aeronautics, hydraulics), the Reynolds number can be as high as several tens of millions, far too high for pure LES models. However, certain regions in the flow can be predicted better with LES than with usual statistical RANS (Reynolds averaged Navier-Stokes) models. These are mainly vortical separated regions as assumed in one of the most popular hybrid models, the hybrid Detached Eddy Simulation model. Here, "hybrid" means that a blending is applied between LES and RANS. An important difference between a real life flow and a wind tunnel or basin is that the turbulence of the flow upstream of each body is not well known.

The development of hybrid models, in particular DES in the litterature, has raised the question of the domain of validity of these models. According to theory, these models should not be applied to flow involving laminar boundary layers (BL). But industrial flows are complex flows and often present regions of laminar BL, regions of fully developed turbulent BL and regions of non-equilibrium vortical BL. It is then mandatory for industrial use that the new hybrid models give a reasonable prediction for all these types of flow. We concentrated on evaluating the behavior of hybrid models for laminar BL and for vortical wakes. While less predictive than pure LES on laminar BL, some hybrid models still give reasonable predictions for rather low Reynolds numbers.

This year, we have developed a new model relying on the hybridation of a DDES model based on a k-ϵ closure with our dynamic VMS model. This model shows improvement in most situations and in particular for laminar flows.

We have also addressed this year a challenging test case, the flow around tandem cylinders with a distance between the cylinders of 12 diameters. The accurate capture of the vortices traveling along this path of 12 diameters requires that the LES filter does not accumulate any dissipation along this trajactory. This is a noticeable property or our DVMS model. Further, the numerics need be as accurate as possible. We use a superconvergent approximation, up to fifth order accurate on Cartesian regions of the computational domain. This combination allowed for an accurate prediction of the drag of the second cylinder. This result has been presented at the workshop ETMM12 [12]

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

- Ecuador and Lemma have a bilateral contract to share the results of Stephen Wornom, Lemma engineer provided to Inria and hosted by Inria under a Inria-Lemma contract.

8. Partnerships and Cooperations

8.1. International Initiatives

8.1.1. Inria International Labs

Ecuador participates in the Joint Laboratory for Exascale Computing (JLESC) together with colleagues at Argonne National Laboratory.

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific events organisation

9.1.1.1. Member of the organizing committees

Laurent Hascoët is on the organizing commitee of the EuroAD Workshops on Algorithmic Differentiation (http://www.autodiff.org).

9.1.2. Scientific Expertise

Alain Dervieux is Scientific Director for the LEMMA company.

9.2. Teaching - Supervision - Juries

9.2.1. Supervision

PhD : Éléonore Gauci, “Goal-oriented metric-based mesh adaptation for unsteady CFD simulations involving moving geometries”, defended december 12, co-advisor A. Dervieux

10. Bibliography

Major publications by the team in recent years

[1] D. GOLDBERG, S. H. K. NARAYANAN, L. HASCOËT, J. UTKE. *An optimized treatment for algorithmic differentiation of an important glaciological fixed-point problem*, in "Geoscientific Model Development", 2016, vol. 9, n^{o} 5, 27 p. , https://hal.inria.fr/hal-01413295

[2] L. HASCOËT. *Adjoints by Automatic Differentiation*, in "Advanced data assimilation for geosciences", Oxford University Press, 2014, https://hal.inria.fr/hal-01109881

[3] L. HASCOËT, M. VÁZQUEZ, B. KOOBUS, A. DERVIEUX. *A Framework for Adjoint-based Shape Design and Error Control*, in "Computational Fluid Dynamics Journal", 2008, vol. 16, n^{o} 4, pp. 454-464

[4] L. HASCOËT, V. PASCUAL. *The Tapenade Automatic Differentiation tool: Principles, Model, and Specification*, in "ACM Transactions On Mathematical Software", 2013, vol. 39, n^{o} 3, http://dx.doi.org/10.1145/2450153.2450158

[5] L. HASCOËT, J. UTKE. *Programming language features, usage patterns, and the efficiency of generated adjoint code*, in "Optimization Methods and Software", 2016, vol. 31, pp. 885 - 903 [*DOI :* 10.1080/10556788.2016.1146269], https://hal.inria.fr/hal-01413332

[6] J. C. HUECKELHEIM, L. HASCOËT, J.-D. MÜLLER. *Algorithmic differentiation of code with multiple context-specific activities*, in "ACM Transactions on Mathematical Software", 2016, https://hal.inria.fr/hal-01413321

Publications of the year

Doctoral Dissertations and Habilitation Theses

[7] É. GAUCI. *Goal-oriented metric-based mesh adaptation for unsteady CFD simulations involving moving geometries*, Université Côte d'Azur, 2018

Articles in International Peer-Reviewed Journals

[8] V. PASCUAL, L. HASCOËT. *Mixed-language automatic differentiation*, in "Optimization Methods and Software", February 2018, vol. 00, pp. 1 - 15 [*DOI :* 10.1080/10556788.2018.1435650], https://hal.inria.fr/hal-01852216

Other Publications

[9] A. BELME, F. ALAUZET, A. DERVIEUX. *An a priori anisotropic Goal-Oriented Error Estimate for Viscous Compressible Flow and Application to Mesh Adaptation*, November 2018, working paper or preprint, https://hal.inria.fr/hal-01927113

[10] A. DERVIEUX, E. GAUCI, L. FRAZZA, A. BELME, A. CARABIAS, A. LOSEILLE, F. ALAUZET. *Mesh-Anpassung für k-genaue Approximationen in CFD*, November 2018, working paper or preprint, https://hal.inria.fr/hal-01927145

[11] E. GAUCI, A. BELME, A. CARABIAS, A. LOSEILLE, F. ALAUZET, A. DERVIEUX. *A priori error-based mesh adaptation in CFD*, December 2018, working paper or preprint, https://hal.inria.fr/hal-01928249

[12] E. ITAM, S. F. WORNOM, B. KOOBUS, A. DERVIEUX. *Combining a DDES model with a dynamic variational multiscale formulation*, November 2018, working paper or preprint, https://hal.inria.fr/hal-01928383

[13] E. ITAM, S. WORNOM, B. KOOBUS, A. DERVIEUX. *A Volume-agglomeration multirate time advancing for high Reynolds number flow simulation*, November 2018, working paper or preprint, https://hal.inria.fr/hal-01928223

References in notes

[14] A. AHO, R. SETHI, J. ULLMAN. *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986

[15] I. ATTALI, V. PASCUAL, C. ROUDET. *A language and an integrated environment for program transformations*, Inria, 1997, n° 3313, http://hal.inria.fr/inria-00073376

[16] B. CHRISTIANSON. *Reverse accumulation and implicit functions*, in "Optimization Methods and Software", 1998, vol. 9, n° 4, pp. 307–322

[17] D. CLÉMENT, J. DESPEYROUX, L. HASCOËT, G. KAHN. *Natural semantics on the computer*, in "Proceedings, France-Japan AI and CS Symposium, ICOT", 1986, pp. 49-89, Also, Information Processing Society of Japan, Technical Memorandum PL-86-6. Also Inria research report # 416, http://hal.inria.fr/inria-00076140

[18] P. COUSOT. *Abstract Interpretation*, in "ACM Computing Surveys", 1996, vol. 28, n° 1, pp. 324-328

[19] B. CREUSILLET, F. IRIGOIN. *Interprocedural Array Region Analyses*, in "International Journal of Parallel Programming", 1996, vol. 24, n° 6, pp. 513–546

[20] J. GILBERT. *Automatic differentiation and iterative processes*, in "Optimization Methods and Software", 1992, vol. 1, pp. 13–21

[21] M.-B. GILES. *Adjoint methods for aeronautical design*, in "Proceedings of the ECCOMAS CFD Conference", 2001

[22] A. GRIEWANK, C. FAURE. *Reduced Gradients and Hessians from Fixed Point Iteration for State Equations*, in "Numerical Algorithms", 2002, vol. 30(2), pp. 113–139

[23] A. GRIEWANK. *On stable piecewise linearization and generalized algorithmic differentiation*, in "Optimization Methods and Software", 2013, vol. 28, n° 6, pp. 1139–1178, http://dx.doi.org/10.1080/10556788.2013.796683

[24] A. GRIEWANK, A. WALTHER. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd, SIAM, Other Titles in Applied Mathematics, 2008

[25] L. HASCOËT. *Transformations automatiques de spécifications sémantiques: application: Un vérificateur de types incremental*, Université de Nice Sophia-Antipolis, 1987

[26] P. HOVLAND, B. MOHAMMADI, C. BISCHOF. *Automatic Differentiation of Navier-Stokes computations*, Argonne National Laboratory, 1997, n° MCS-P687-0997

[27] E. LAROUR, J. UTKE, B. CSATHO, A. SCHENK, H. SEROUSSI, M. MORLIGHEM, E. RIGNOT, N. SCHLEGEL, 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", 2014, vol. 8, n° 6, pp. 2335-2351 [*DOI :* 10.5194/TC-8-2335-2014], http://www.the-cryosphere.net/8/2335/2014/

[28] F.-X. LE DIMET, O. TALAGRAND. *Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects*, in "Tellus", 1986, vol. 38A, pp. 97-110

[29] B. MOHAMMADI. *Practical application to fluid flows of automatic differentiation for design problems*, in "Von Karman Lecture Series", 1997

[30] N. ROSTAING. *Différentiation Automatique: application à un problème d'optimisation en météorologie*, université de Nice Sophia-Antipolis, 1993

[31] R. RUGINA, 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