INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team tropics

Program transformations for scientific computing

Sophia Antipolis - Méditerranée

Theme : Computational models and simulation

Table of contents

1. Team

Research Scientists

Laurent Hascoët [Senior Researcher, Team leader, HdR]
Valérie Pascual [Junior Researcher]
Alain Dervieux [Senior Researcher, HdR]

Faculty Member

Bruno Koobus [Université Montpellier 2, HdR]

External Collaborator

Stephen Wornom [Lemma Company]

PhD Students

Anca Belme
Hubert Alcin
Alexandre Carabias

Administrative Assistant

Claire Senica

2. Overall Objectives

2.1. Overall Objectives

The TROPICS team studies Automatic Differentiation (AD) of algorithms and programs. We work at the junction of two research domains:

- **AD theory:** On the one hand, we study software engineering techniques, to analyze and transform programs mechanically. Automatic 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 so-called *reverse* or *adjoint* mode of AD, a complex transformation that yields gradients for optimization at a remarkably low cost.
- **AD application to Scientific Computing:** On the other hand, we study application of the adjoint mode of AD to e.g. Computational Fluid Dynamics. We adapt the strategies used in Scientific Computing in order to take full advantage of AD. This work is applied to several real-size applications.

Each aspect of our work benefit to the other. We want to produce AD code that can compete with hand-written sensitivity and adjoint programs that are used in the industry. We implement our algorithms into our tool TAPENADE, which is now one of the most popular AD tools.

Our research directions are :

- Modern numerical methods for finite elements or finite differences : multigrid methods, mesh adaptation.
- Optimal shape design or optimal control in fluid dynamics for steady and unsteady simulations. Higher-order derivatives needed by robust optimization.
- Automatic Differentiation : AD-specific static data-flow analysis, strategies to reduce runtime and memory consumption of the reverse mode in the case of very large codes. Improved models for reverse AD, in particular coping with message-passing parallellism.

3. Scientific Foundations

3.1. Automatic Differentiation

Participants: Laurent Hascoët, Valérie Pascual.

Glossary

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

adjoint model Mathematical manipulation of the Partial Derivative 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 the reverse mode of AD, that trades duplicate execution of a part of the program to save some memory space that was used to save intermediate results. Checkpointing a code fragment amounts to running this fragment without any storage of intermediate values, thus saving memory space. Later, when such an intermediate value is required, the fragment is run a second time to obtain the required values.

Automatic or Algorithmic Differentiation (AD) differentiates *programs*. An AD tool takes as input a source computer program P that, given a vector argument $X \in I\!R^n$, computes some vector function $Y = F(X) \in I\!R^m$. The AD tool generates a new source program P' that, given the argument X, computes some derivatives of F. The resulting P' reuses the control of P. Therefore, strictly speaking, P' evaluates F' only piecewise. Experience shows that this is reasonable in most cases. Going further is still an open research problem.

For any given control, P is equivalent to a sequence of instructions, which is identified with a composition of vector functions. Thus, if

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

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 chain rule gives the Jacobian of F

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

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

In practice, the Jacobian $F'(X)$ is often far too expensive to compute and store. Fortunately, most problems are solved using only some projections of $F'(X)$. For example, one may need only *sensitivities*, which are $F'(X).\dot{X}$ for a given direction $\dot{X}$ in the input space. Using equation (2), sensitivity is

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

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

However in optimization, data assimilation [39], adjoint problems [33], or inverse problems, the appropriate derivative is the *gradient* $F'^*(X).\overline{Y}$, where F' has been transposed. Using equation (2), the gradient is

$$F'^*(X).\overline{Y} = f_1'^*(X_0).f_2'^*(X_1).\ \cdots\ .f_{p-1}'^*(X_{p-2}).f_p'^*(X_{p-1}).\overline{Y}, \tag{4}$$

which is most efficiently computed from right to left, because matrix×vector products are cheaper than matrix×matrix products. This is the principle of the *reverse mode* of AD.

This turns out to make a very efficient program, at least theoretically [36]. The computation time required for the gradient is only a small multiple of the run-time of P. It is independent from the number of parameters n. In contrast, notice that computing the same gradient with the *tangent mode* would require running the tangent differentiated program n times.

However, we observe that 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 the reverse mode 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 communication. We regularly extend our models and tools to handle these new constructs.

3.2. Static Analysis and Transformation of programs

Participants: Laurent Hascoët, Valérie Pascual.

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, contain each a list of instructions to be executed in sequence, and whose arcs 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 simultaneously, 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 whether a variable is initialized or not, at any location in the source program.

data dependence analysis Program analysis that studies the itinerary of values during program execution, from the place where a value is generated to the places where it is used, and finally to the place where it is overwritten. The collection of all these itineraries is often stored as a *data dependence graph*, and data flow analysis most often rely on this graph.

data dependence graph Directed graph that relates accesses to program variables, from the write access that defines a new value to the read accesses that use this value, and conversely from the read accesses to the write access that overwrites this value. Dependences express a partial order between operations, that must be preserved to preserve the program's result.

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 use sophisticated analysis [25] to improve the quality of the produced code. These tools share their technological basis. More importantly, there are common mathematical models to specify and analyze them.

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* and *back-ends*. But one can go further: as abstraction goes on, the internal representation becomes more language independent, and semantic constructs can be unified. Analysis can then concentrate on the semantics of a small set of constructs. We advocate an internal representation composed of three levels.

- At the top level is the *call graph*, whose nodes are modules and procedures. Arrows relate nodes that call or import one another. Recursion leads to cycles.
- At the middle level is the *flow graph*, one per procedure. It captures the control flow between atomic instructions.
- At the lowest level are abstract *syntax trees* for the individual atomic instructions. Semantic transformations can benefit from the representation of expressions as directed acyclic graphs, sharing common sub-expressions.

At each level are associated symbol tables, that are nested to capture the notion of visibility scope.

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 [27], [37], [26]. But many analyses are more complex, and better defined on graphs than on trees. This is the case for *data-flow analyses*, that look for run-time properties of variables. Since flow graphs are cyclic, these global analyses generally require an iterative resolution. *Data flow equations* is a practical formalism to describe data-flow analyses. Another formalism is described in [28], which is more precise because it can distinguish separate *instances* of instructions. However it is still based on trees, and its cost forbids application to large codes. *Abstract Interpretation* [29] 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. In addition to the very theoretical limit of *undecidability*, there are practical limitations to how much information one can infer from programs that use arrays [43], [30] or pointers. In general, conservative *over-approximations* are always made that lead to derivative code that is less efficient than possibly achievable.

3.3. Automatic Differentiation and Scientific Computing

Participants: Alain Dervieux, Laurent Hascoët, Bruno Koobus.

Glossary

linearization In Scientific Computing, the mathematical model often consists of Partial Derivative 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 Derivative Equations that define some characteristics of a system with respect to some input parameters. Consider one particular scalar characteristic. Its sensitivity, (or gradient) with respect to the input parameters can be defined as the solution 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 now provides reliable simulations of very complex systems. For example it is now possible to simulate the 3D air flow around a plane that captures the physical phenomena of shocks and turbulence. The next step appears to be optimization. Optimization is one degree higher in complexity, because it repeatedly simulates, evaluates directions of optimization and applies optimization steps, until an optimum is reached. We focus on gradient-based optimization. We are aware of the problems due to local minima, which require more global optimization methods. Still, gradient-based optimization is necessary, even coupled with global methods, to efficiently reach the bottom of the nearest local minimum.

We investigate several approaches to obtain the gradient. There are actually two extreme approaches:

- One can write an *adjoint system* of mathematical equations, then discretize it and program it by hand. This is mathematically sound [33], but very costly in development time. It also does not produce an exact gradient of the discrete function, and this can be a problem if using optimization methods based on descent directions.
- One can apply reverse AD (*cf* 3.1) on the program that discretizes and solves the direct system. This gives in fact the adjoint of the discrete function computed by the program. Theoretical results [32] 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 [38], [41] to reduce memory usage.

We study approaches between these extremes. If for instance the model is steady, one can use the iterated states in the direct order [34], or one can use only the fully converged final state. Since these mixed approaches can also be error-prone, we advocate incorporating them into the AD model and into the AD tools.

4. Application Domains

4.1. Panorama

Automatic Differentiation of programs gives sensitivities or gradients, that are useful for many types of applications:

- 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,
- mesh adaptation and mesh optimization with gradients or adjoints,
- equation solving with the Newton method,
- sensitivity analysis, propagation of truncation errors.

These applications require an AD tool that differentiates programs written in classical imperative languages, FORTRAN77, FORTRAN95, C, or C++.

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. From this flow, it computes an optimization criterion, such as the lift of an aircraft. To optimize the criterion by a gradient descent, one needs the gradient of the output criterion with respect to all the inputs, and possibly additional gradients when there are constraints. The reverse mode of AD is a promising 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 the altitude and speed of the surface.

One particular case of inverse problems is *data assimilation* [39] 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 largely unknown. Only some measures at arbitrary places and times are available. A good initial state is found by solving a least squares problem between the measures 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 [42]. Figure 1 shows an example of a data assimilation exercise using the oceanography code OPA [40] and its AD adjoint code produced by TAPENADE.

The special case of *4Dvar* data assimilation is particularly challenging. The 4^{th} dimension in "4D" is time, as available measures 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 grows in complexity, 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 is sometimes too expensive, in particular in the context of real time. When one wants to simulate the reaction of this complex system to small perturbations around a fixed set of parameters, there is a very efficient approximate solution: 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. Software

5.1. TAPENADE

Participants: Laurent Hascoët [correspondant], Valérie Pascual.

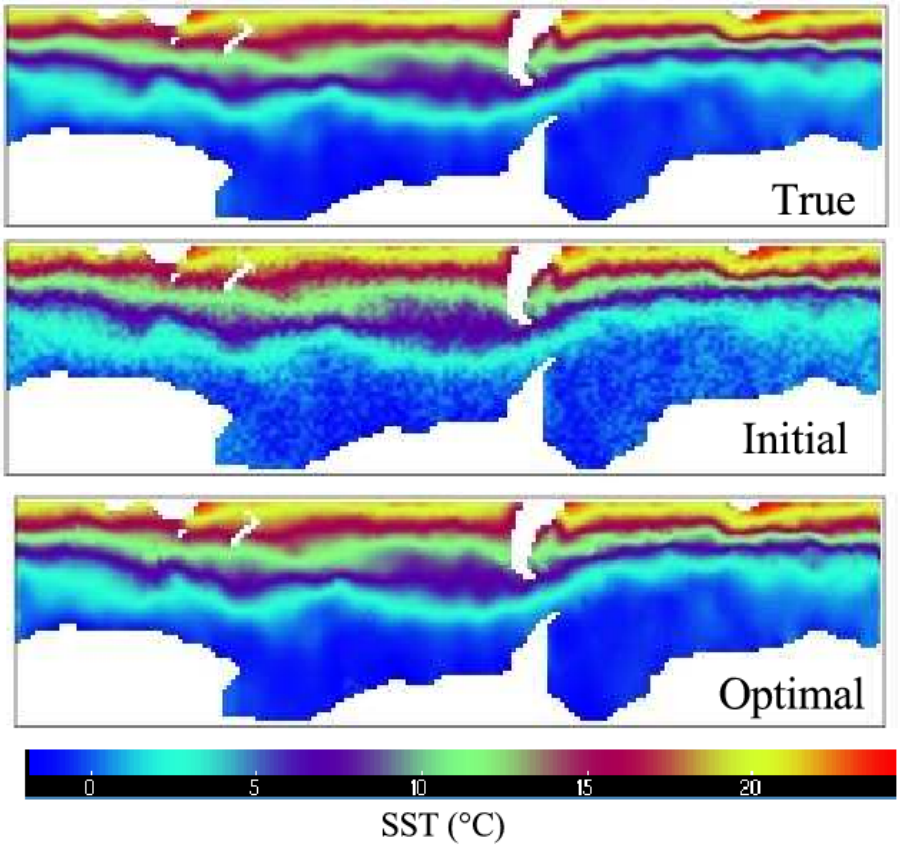

Figure 1. Twin experiment using the adjoint of OPA. We add random noise to a simulation of the ocean state around the Antarctic, and we remove this noise by minimizing the discrepancy with the physical model

- ACM: D.3.4 Compilers, G.1.0 Numerical algorithms, G.1.4 Automatic differentiation, I.1.2 Analysis of algorithms
- AMS: 65K10, 68N20
- Keywords: automatic differentiation, adjoint, gradient, optimisation, inverse problems, static analysis, data-flow analysis, compilation
- APP registration: IDDN.FR.001.040038.002.S.P.2002.000.10600

TAPENADE is the Automatic Differentiation tool developed by the TROPICS team. TAPENADE implements the results of our research about models and static analyses for AD. To promote usage of AD in the scientific computation world, including the industry, we constantly maintain TAPENADE to meet the demands of our end-users. TAPENADE is writtten in JAVA and can be downloaded and installed on most architectures. Alternatively, it can be used as a web server. All information is available from the team's web page
http://www-sop.inria.fr/tropics/

TAPENADE differentiates computer programs according to the model described in section 3.1. It supports three modes of differentiation: tangent, vector (i.e. multi-directional) tangent, and reverse. Higher-order derivatives can be obtained through repeated application of tangent AD on tangent and/or reverse AD. TAPENADE accepts programs written in FORTRAN77, FORTRAN95, and C. Thanks to the language-independent internal representation of programs, a single TAPENADE kernel is used and every further development in TAPENADE benefits to differentiation of each input language.

TAPENADE performs sophisticated data-flow analysis on the complete source program to produce an efficient differentiated code. All these data-flow analysis are both flow-sensitive and context-sensitive. Classical analysis, not specific to AD, include Type-Checking, Read-Write analysis, and Pointer analysis. AD-specific analysis include:

- **Activity analysis:** This detects variables whose derivative is either null or useless. The interest is to reduce the number of derivative instructions.
- **Adjoint Liveness analysis:** This detects the source statements that are not needed for the computation of derivatives. The interest is to reduce the number of source statements that are copied into the derivative code.
- **TBR analysis:** In reverse mode, this finds the smallest sets of source variables that need to be preserved for use in the derivative statements. The interest is to reduce the memory consumption of the reverse mode.

This year, in addition to the usual debugging activity in response to reports from our end-users, we have included the following major features:

- A storage-recomputation tradeoff that reduces the memory consumption in reverse mode at the cost of repeated execution of selected simple statements of the source program.
- A finer control on the checkpointing mechanism, that lets the end-user define portions of procedures on which checkpointing must be applied.
- The automatic setup by TAPENADE of the machinery for optimal (binomial) checkpointing [35] on iterative loops.
- (still under development) The taking into account of parallel communication calls along all the chain of the TAPENADE tool.

TAPENADE is not open-source. Academic usage is free. Industrial or commercial usage require a paying license, as detailled on the team's web page. Several industrial companies have purchased and renewed an industrial license for TAPENADE. This year's new custommers are Exxon-Mobil, BASF, and UTC. The software has been downloaded several hundred times, and the web tool served several thousands of true connections (not robots). The tapenade-users mailing list has reached one hundred registered users.

6. New Results

6.1. Automatic Differentiation and parallel codes

Participants: Valérie Pascual, Laurent Hascoët, Jean Utke [Argonne National Lab. (Illinois, USA)], Uwe Naumann [RWTH Aachen University (Germany)].

This research is an ongoing joint work between three teams working on AD. We study differentiation in reverse mode of programs that contain MPI communication calls. Instead of the commonly used approach that encapsulates the MPI calls into black-box subroutines that will be differentiated by hand, we are looking for a native differentiation of the MPI calls by the AD tool.

One issue is to reduce the large variability of the available MPI calls and parameters to a smaller number of elementary concepts. We then address the basic question of `send`s and `recv`s, that may be blocking or nonblocking, individual or collective, and so on. Essentially the adjoint of a `send` is a `recv`, and vice-versa, but the possibility of nonblocking `isend`'s and `irecv`'s requires more subtlety.

This year, we focused on the adaption of the tool's static analysis to programs with parallel communication. It requires conceptual development to integrate this communication into our framework of flow-sensitive and context-sensitive data-flow analysis. As an experiment, we started to implement these new concepts into TAPENADE's data-flow analysis. Results are still preliminary, but the approach correctly captures the influence of communication on the data-flow. In particular, this approach retains a high level of flow-sensitivity.

Consistently with our general choices, we focus on AD tools based on program transformation. Therefore we adapt the general ideas on differentiation of parallel communication to program transformation, and this is why we need to adapt the data-flow analysis components. On the other hand, we closely follow the developments of these general ideas for operator-overloading AD tools, which require a more complex definition of the overloaded communication primitives [22].

6.2. Combined Storage and Recomputation for Data-Flow reversal

Participant: Laurent Hascoët.

The Data-Flow reversal inherent to the reverse mode of AD is bound to have a cost in memory space or in duplicate computations.

Last year, we started to implement a practical strategy to replace some Storage with cheap Recomputation. This strategy only picks some "low-hanging fruit", as it considers only recomputation that obeys some simple data-flow properties.

This year, we continued this development. Successive refinements make this strategy more and more complex, and error-prone. A proof of correction becomes necessary, that takes into account all possible interactions between this strategy and the data-flow analysis that it uses and influences. We are building such a proof of correctness, that still needs to be refined and simplified before it is published.

In the future, we plan to lift more limitations of this strategy. One goal is to encompass the extreme "Recompute-All" strategy that is implemented in the TAF tool [31], with its optimizations ("Efficient Recomputation Algorithm"). Another goal is to use this strategy as a framework to get closer to an optimum between storage and recomputation.

6.3. Resolution of linearised systems

Participants: Hubert Alcin, Olivier Allain [Lemma], Anca Belme, Marianna Braza [IMF-Toulouse], Alexandre Carabias, Alain Dervieux, Bruno Koobus [Université Montpellier 2], Carine Moussaed [Université Montpellier 2], Hilde Ouvrard [IMF-Toulouse], Stephen Wornom [Lemma].

The interaction between the sophisticated solution algorithm inside a program and the Automatic Differentiation of the program is a non-trivial issue. An iterative algorithm generally does not store the successive updates of the iterated solution vector. Furthermore, a modern iterative solution algorithm involves several nonlinear processes, like in:

- the evaluation of an optimal step, which results at least from a homographic function of the unknown,
- the orthonormalisation of the updates (Gram-Schmidt method, Hessenberg method).

Applying reverse AD to the iterative solution algorithm produces a *linearised iterative algorithm* which is transposed and therefore follows a reverse order, with exactly the same number of iterations, and needing exactly each of the iterated state solution vectors. This effect is considerably amplified in the case of the numerical simulation of unsteady phenomena with implicit numerical schemes. For example, the simulation of high Reynolds turbulent flows by a Large Eddy Simulation (LES) requires hundreds of thousands time steps, each of them involving a modern iterative solution algorithm.

In the ECINADS ANR project, we design more efficient solution algorithms and we examine the questions risen by their reverse differentiation. The application domain is the computation of high Reynolds turbulent flows with LES and hybrid RANS-LES models. The efficiency will be evaluated through the practical scalability on a large number of processors. This efficiency criterion also extends to the scalability of the reverse/adjoint algorithm. ECINADS also addresses the scalable solution of new approximations. ECINADS associates the university of Montpellier 2, the Institut de Mécanique des fluides de Toulouse and Lemma company. The kick off meeting of ECINADS was held at end of 2009.

Hubert Alcin started his PhD in october 2009, with advisors Olivier Allain and Alain Dervieux. He is studying coarse grid methods for domain decomposition for the Poisson solver used in the projection step for an incompressible model. The ingredients are Deflation or Balancing methods for introducing coarse grids and an additive Schwarz algorithm. Two approaches are used for building a coarse grid basis, either with the characteristic functions of the partition or with smoothed version of them. Numerical experiments show that using the smooth basis produces a better scalability. These results were discussed in a ECINADS seminar and at the ECINADS period review. Carinne Moussaed et Bruno Koobus in Montpellier started an extension to models for compressible flows. Alexandre Carabias started the study of higher order numerical advection schemes, with advisors Oliver Allain and Alain Dervieux. He is extending a scheme introduced by Hilde Ouvrard and Bruno Koobus.

6.4. Perturbation Methods

Participants: Anca Belme, Massimiliano Martinelli [Universitá di Pavia], Alain Dervieux, Laurent Hascoët, Régis Duvigneau [OPALE team].

In the context of the European project NODESIM-CFD, the contribution of Tropics involved mainly the production of second derivative code through repeated application of Automatic Differentiation. Three strategies can be applied to obtain (elements of) the Hessian matrix, named Tangent-on-Tangent (ToT), Tangent-on-Reverse (ToR), and Reverse-on-Tangent (RoT). The subject of correction of approximation errors is also a contribution to NODESIM-CFD and an important application of TAPENADE. We investigated the two types of correctors, by direct linearisation and Defect Correction, or by the adjoint-based functional correction. These contribution were reported in the "Guide for Uncertainty Management in CFD" written in collaboration with NUMECA and Vrije Universiteit Brussels and in [15].

6.5. Control of approximation errors

Participants: Frédéric Alauzet [GAMMA team, INRIA-Rocquencourt], Olivier Allain [Lemma], Anca Belme, Alain Dervieux, Damien Guegan [Lemma], Adrien Loseille [GAMMA team, INRIA-Rocquencourt].

This is a joint research between INRIA teams GAMMA (Rocquencourt), TROPICS, and PUMAS. Roughly speaking, GAMMA brings mesh and approximation expertise, TROPICS contributes to adjoint methods, and CFD applications are developed in the context of PUMAS.

The resolution of the optimum problem using the innovative approach of an AD-generated adjoint can be used in a slightly different context than optimal shape design namely, mesh adaptation. This will be possible if we can map the mesh adaptation problem into a differentiable optimal control problem. To this end, we have introduced a new methodology that consists in stating the mesh adaptation problem in a purely functional form: the mesh is reduced to a continuous property of the computational domain: the continuous metric. We minimize a continuous model of the error resulting from that metric. Thus the problem of searching an adapted mesh is transformed into the search of an optimal metric.

In the case of mesh interpolation minimization, the optimum is given by a close formula and gives access to a complete theory demonstrating that second order accuracy can be obtained on discontinuous field approximation, [13]. In the case of adaptation for Partial Differential Equations such as the Euler model, we need an adjoint state that we obtain with TAPENADE. We end up with a minimisation problem for the metric which in turn is solved analytically [14], [21], [20]. During 2010, the extension to unsteady problems has been started, see [18], [19].

7. Dissemination

7.1. Animation of the scientific community

- TROPICS participates in the project EVA-Flo: "Evaluation et Validation Automatique pour le calcul FLOttant", which is an ANR project accepted in 2007, and whose main contractor in ENS Lyon (Nathalie Revol). Laurent Hascoët attended an EVA-Flo project meeting in Perpignan, France (may 20-21).
- Hubert Alcin presented his results on coarse grid methods in CANUM 2010 in Carcans (may) and at the ECINADS seminar in Sophia-Antipolis (october).
- Laurent Hascoët is on the organizing commitee of the European Workshops on Automatic Differentiation. He attented this year's workshops in Paderborn, Germany (june 3-4) and Cranfield, UK (december 9).
- Laurent Hascoët is a member of the internal "CDT" committee at INRIA Sophia-Antipolis ("Comité Développement Technologique").
- Anca Belme presented a talk on "Goal-oriented anisotropic mesh adaptation for unsteady flows" at ECCOMAS 2010, Lisbon.
- Alain Dervieux presented a talk on "Fully anisotropic goal-oriented mesh adaptation: 3D anisotropic mesh adaptation for functional outputs" at ECCM2010, Paris.
- Alain Dervieux presented a lecture on " Indicateurs de raffinement et adaptation de maillage en simulation numérique pour la mécanique des fluides" at Collège Polytechnique.
- Laurent Hascoët presented the team's research on AD at CNAM Paris ("Conservatoire National des Arts et Métiers") (september 22).
- The team hosted a scientific seminar to celebrate the 60^{th} birthday of Andreas Griewank (HU Berlin) (april 8-9). A smaller seminar was organized with the Tropics team on april 7, with Andreas Griewank, Jorge Moré (Argonne), R. Baker Kearfott (Louisiana University) and Trond Steihaug (Bergen University).
- The team organized a workshop of the ECINADS ANR project at INRIA Sophia-Antipolis (october 27-28).
- The team participated in the European STREP project NODESIM (Non-Deterministic Simulation for CFD-based design methodologies), driven by Numeca (Belgium) ended this year. TROPICS and OPALE contributed to application of AD to build reduced models using first and second derivatives. We design robust optimization strategies, and correctors for approximation errors.

- The team is coordinator of the ANR project ECINADS, with PUMAS team, university Montpellier 2, Institut de mécanique des Fluides de Toulouse and the Lemma company in Sophia-Antipolis. ECINADS concentrates on solution algorithms for state and adjoint systems in CFD.
- The team's PhD students organized a joint seminar day with the PhD students of the INRIA team NACHOS and the CEMEF (Ecole des mines de Paris) (june 7).
- Alain Dervieux was in the PhD jury of Julien Montagnier (Lyon), of Ludovic Martin and Guillaume Barbut (university of Toulouse).

7.2. Teaching

- Anca Belme gives lectures at Université de Nice on Numerical Algorithms (48 hours)
- Hubert Alcin gives lectures to 3rd year students at Université de Nice.

8. Bibliography

Major publications by the team in recent years

[1] F. COURTY, A. DERVIEUX. *Multilevel functional Preconditioning for shape optimisation*, in "International Journal of CFD", 2006, vol. 20, n^{o} 7, p. 481-490.

[2] F. COURTY, A. DERVIEUX, B. KOOBUS, L. HASCOËT. *Reverse automatic differentiation for optimum design: from adjoint state assembly to gradient computation*, in "Optimization Methods and Software", 2003, vol. 18, n^{o} 5, p. 615-627.

[3] B. DAUVERGNE, L. HASCOËT. *The Data-Flow Equations of Checkpointing in reverse Automatic Differentiation*, in "International Conference on Computational Science, ICCS 2006, Reading, UK", 2006.

[4] A. GRIEWANK. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, Frontiers in Applied Mathematics, 2000.

[5] L. HASCOËT, M. ARAYA-POLO. *The Adjoint Data-Flow Analyses: Formalization, Properties, and Applications*, in "Automatic Differentiation: Applications, Theory, and Tools", H. M. BÜCKER, G. CORLISS, P. HOVLAND, U. NAUMANN, B. NORRIS (editors), Lecture Notes in Computational Science and Engineering, Springer, 2005.

[6] L. HASCOËT, S. FIDANOVA, C. HELD. *Adjoining Independent Computations*, in "Automatic Differentiation of Algorithms: From Simulation to Optimization", New York, NY, G. CORLISS, C. FAURE, A. GRIEWANK, L. HASCOËT, U. NAUMANN (editors), Computer and Information Science, Springer, New York, NY, 2001, chap. 35, p. 299-304.

[7] L. HASCOËT, U. NAUMANN, V. PASCUAL. *"To Be Recorded" Analysis in Reverse-Mode Automatic Differentiation*, in "Future Generation Computer Systems", 2004, vol. 21, n^{o} 8.

[8] L. HASCOËT, J. UTKE, U. NAUMANN. *Cheaper Adjoints by Reversing Address Computations*, in "Scientific Programming", 2008, vol. 16, n^{o} 1, p. 81–92.

[9] 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, p. 454-464.

[10] M. VÁZQUEZ, A. DERVIEUX, B. KOOBUS. *Multilevel optimization of a supersonic aircraft*, in "Finite Elements in Analysis and Design", 2004, vol. 40, p. 2101-2124.

Publications of the year

Articles in International Peer-Reviewed Journal

[11] R. BOURGUET, M. BRAZA, A. DERVIEUX. *Reduced order modeling of transonic flows around an airfoil*, in "Journal of Computational Physics", 2010, to appear.

[12] A. DERVIEUX, C. FARHAT, B. KOOBUS, M. VÁZQUEZ. *Total energy conservation in ALE schemes for compressible flows*, in "European Journal of Computational Mechanics", 2010, vol. 19:4, p. 337-363.

[13] D. GUÉGAN, O. ALLAIN, A. DERVIEUX, F. ALAUZET. *A L-infinity-Lp mesh adaptive method for computing unsteady bi-fluid flows*, in "Int. J. Numerical Methods in Engineering", 2010, to appear.

[14] A. LOSEILLE, A. DERVIEUX, F. ALAUZET. *Fully anisotropic goal-oriented mesh adaptation for 3D steady Euler equations*, in "Journal of Computational Physics", 2010, vol. 229, p. 2866-2897.

[15] M. MARTINELLI, A. DERVIEUX, L. HASCOËT, V. PASCUAL, A. BELME. *AD-based perturbation methods for uncertainties and errors*, in "International Journal of Engineering Systems Modelling and Simulation", 2010, vol. 2, n^{o} 1/2, p. 65-74.

[16] H. OUVRARD, B. KOOBUS, A. DERVIEUX, M.-V. SALVETTI. *Classical and Variational Multiscale Large-Eddy Simulations of the Flow around a Circular Cylinder on Unstructured Grids*, in "Computer and Fluids", 2010, vol. 39, p. 1083-1094.

International Peer-Reviewed Conference/Proceedings

[17] A. BELME, A. DERVIEUX, F. ALAUZET. *Fully anisotropic goal-oriented mesh adaptation for unsteady flows*, in "Proceedings of the ECCOMAS-CFD 2010 conference", 2010.

[18] A. BELME, A. DERVIEUX, F. ALAUZET. *Mesh-adaptive computation of linear and non-linear acoustics*, in "Proceedings of Trilateral Seminar on Computational Experiment in Aeroacoustics", Svetlogorsk, Russia, 2010.

[19] A. BELME, A. DERVIEUX, B. KOOBUS, S. WORNOM, M.-V. SALVETTI. *Application of hybrid and vms-les turbulent models to aerodynamic simulations*, in "Proceedings of ICAS Conference", Nice, 2010, to appear.

[20] A. LOSEILLE, A. DERVIEUX, F. ALAUZET. *A 3D goal-oriented anisotropic mesh adaptation applied to inviscid flows in aeronautics*, in "Proceedings of 48th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2010-1067", 2010.

[21] A. LOSEILLE, A. DERVIEUX, F. ALAUZET. *Fully anisotropic goal-oriented mesh adaptation :3D anisotropic mesh adaptation for functional outputs*, in "Proceedings of IV European Conference on Computational Mechanics,ECCM2010", 2010.

[22] M. SCHANEN, U. NAUMANN, L. HASCOËT, J. UTKE. *Interpretative Adjoints for Numerical Simulation Codes using MPI*, in "Proceedings of the 10th International Conference on Computational Science, ICCS'2010", 2010.

Scientific Books (or Scientific Book chapters)

[23] B. KOOBUS, F. ALAUZET, A. DERVIEUX. *Numerical algorithms for unstructured meshes*, F. MAGOULES (editor), CRC Press, 2010, to appear.

[24] H. OUVRARD, M.-V. SALVETTI, S. CAMARRI, S. WORNOM, A. DERVIEUX, B. KOOBUS. *LES, variational multiscale LES and hybrid models*, F. MAGOULES (editor), CRC Press, 2010, to appear.

References in notes

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

[26] I. ATTALI, V. PASCUAL, C. ROUDET. *A language and an integrated environment for program transformations*, INRIA, 1997, n^{o} 3313, http://hal.inria.fr/inria-00073376.

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

[28] J.-F. COLLARD. *Reasoning about program transformations*, Springer, 2002.

[29] P. COUSOT. *Abstract Interpretation*, in "ACM Computing Surveys", 1996, vol. 28, n^{o} 1, p. 324-328.

[30] B. CREUSILLET, F. IRIGOIN. *Interprocedural Array Region Analyses*, in "International Journal of Parallel Programming", 1996, vol. 24, n^{o} 6, p. 513–546.

[31] R. GIERING. *Tangent linear and Adjoint Model Compiler , Users manual 1.2*, 1997, http://www.autodiff.com/tamc.

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

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

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

[35] A. GRIEWANK. *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation*, in "Optimization Methods and Software", 1992, vol. 1, p. 35-54.

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

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

[38] P. HOVLAND, B. MOHAMMADI, C. BISCHOF. *Automatic Differentiation of Navier-Stokes computations*, Argonne National Laboratory, 1997, n^{o} MCS-P687-0997.

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

[40] G. MADEC, P. DELECLUSE, M. IMBARD, C. LEVY. *OPA8.1 ocean general circulation model reference manual*, Pole de Modelisation, IPSL, 1998.

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

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

[43] 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.