INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team tropics

Transformations et Outils Informatiques pour le Calcul Scientifique

Sophia Antipolis - Méditerranée

Theme : Computational models and simulation

Table of contents

1. Team

Research Scientist
Laurent Hascoët [Team leader, HdR]
Valérie Pascual
Alain Dervieux [Habilite]

Faculty Member
Bruno Koobus [Université Montpellier 2, HdR]

External Collaborator
Stephen Wornom [Lemma Company]

PhD Student
Anca Belme
Hubert Alcin [Since 01/10]

Administrative Assistant
Christine Faber

Other
Francesco Ghini [Internship from 01/09 to 31/12]
Karim Hossen [Internship from 14/04 to 14/09]

2. Overall Objectives

2.1. Overall Objectives

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

- **AD theory:** On the one hand, we study software engineering techniques, to analyze and transform programs semi-automatically. Our application is Automatic Differentiation (AD). AD transforms a program `P` that computes a function F, into a program `P'` that computes some derivatives of F, analytically. We put a particular emphasis on the *reverse mode* of AD (sometimes called *adjoint mode*), which yields gradients for optimization at a remarkably low cost. The reverse mode of AD requires carefully crafted algorithms.
- **AD application to Scientific Computing:** On the other hand, we study the application of AD, and particularly of the adjoint method, to e.g. Computational Fluid Dynamics. This involves adapting of the strategies used in Scientific Computing, in order to take full advantage of AD. This work is applied to several real-size applications.

The second aspect of our work is thus at the same time the motivation and the application domain of the first aspect. Our objective is to automatically produce AD code that can compete with the hand-written sensitivity and adjoint programs which exist in the industry. We implement our ideas and algorithms into the tool TAPENADE, which is developed and maintained by the team, and which has become one of the most popular AD tools. TAPENADE is available as a web service, and alternatively a version can be downloaded from our web server. Practical details can be found in section 5.1.

Our research directions are :

- Modern numerical methods for finite elements or finite differences: multigrid methods, mesh adaptation.
- Optimal shape design or optimal control in the context of fluid dynamics: This involves optimization of nonsteady processes and computation of higher-order derivatives e.g. for robust optimization.

- Automatic Differentiation : improve the AD models and implement the program static analysis that they require. Devise specific AD strategies for frequent numerical algorithms. Reduce runtime and memory consumption of the reverse mode, study storage/recomputation strategies for very large codes.
- Common tools for program analysis and transformation: adequate internal representation, Call Graphs, Flow Graphs, Data-Dependence Graphs.

3. Scientific Foundations

3.1. Automatic Differentiation

Participants: Laurent Hascoët, Valérie Pascual.

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 $X \in \mathbb{R}^n$, computes some vector function $Y = F(X) \in \mathbb{R}^m$. The AD tool generates a new source program that, given the argument X, computes some derivatives of F. In short, AD first assumes that P represents all its possible run-time sequences of instructions, and it will in fact differentiate these sequences. Therefore, the *control* of P is put aside temporarily, and AD will simply reproduce this control into the differentiated program. In other words, P is differentiated only piecewise. Experience shows that this is reasonable in most cases, and going further is still an open research problem. Then, any sequence of instructions is identified with a composition of vector functions. Thus, for a given control:

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

where each f_k is the elementary function implemented by instruction I_k. Finally, AD simply applies the chain rule to obtain derivatives of F. Let us call X_k the values of all variables after each instruction I_k, i.e. $X_0 = X$ and $X_k = f_k(X_{k-1})$. The chain rule gives the Jacobian F' 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 translated back into a sequence of instructions I'_k, and these sequences inserted back into the control of P, yielding program P'. This can be generalized to higher level derivatives, Taylor series, etc.

In practice, the above Jacobian $F'(X)$ is often far too expensive to compute and store. Notice for instance that equation (2) repeatedly multiplies matrices, whose size is of the order of $m \times n$. Moreover, 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 *tangent mode* of AD, which is the most straightforward, of course available in TAPENADE.

However in optimization, data assimilation [34], adjoint problems [29], or inverse problems, the appropriate derivative is the *gradient* $F'^*(X).\overline{Y}$. 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 so much 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 [31]. 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 $f'^*_{k+1}(X_k)$. This is the main problem of the reverse mode. There are two strategies for addressing it:

- **Recompute All (RA):** the X_k is recomputed when needed, restarting P on input X_0 until instruction I_k. The TAF [27] tool uses this strategy. Brute-force RA strategy has a quadratic time cost with respect to the total number of run-time instructions p.
- **Store All (SA):** the X_k are restored from a stack when needed. This stack is filled during a preliminary run of P, that additionally stores variables on the stack just before they are overwritten. The ADIFOR [22] and TAPENADE tools use this strategy. Brute-force SA strategy has a linear memory cost with respect to p.

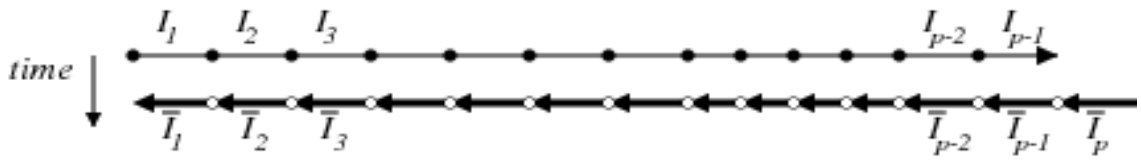

Figure 1. The "Store-All" tactic

Both RA and SA strategies need a special storage/recomputation trade-off in order to be really profitable, and this makes them become very similar. This trade-off is called *checkpointing*. Since TAPENADE uses the SA strategy, let us describe checkpointing in this context. The plain SA strategy applied to instructions I_1 to I_p builds the differentiated program sketched on figure 1, where an initial "forward sweep" runs the original program and stores intermediate values (black dots), and is followed by a "backward sweep" that computes the derivatives in the reverse order, using the stored values when necessary (white dots). Checkpointing a fragment **Ckp** of the program is illustrated on figure 2. During the forward sweep, no value is stored while in **Ckp**. Later, when the backward sweep needs values from **Ckp**, the fragment is run again, this time with storage. One can see that the maximum storage space is grossly divided by 2. This also requires some extra memorization (a "snapshot"), to restore the initial context of **Ckp**. This snapshot is shown on figure 2 by slightly bigger black and white dots.

*Figure 2. Checkpointing **Ckp** with the "Store-All" tactic*

Checkpoints can be nested. In that case, a clever choice of checkpoints can make both the memory size and the extra recomputations grow only like the logarithm of the size of the program.

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, 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 [20] 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. In particular, it is desirable that the optimization and code generation phases be independent from the particular input programming language. This can generally be achieved through separate *front-ends*, that produce an internal language-independent representation of the program, generally an abstract syntax tree. For example, compilers like `gcc` for C and `g77` for FORTRAN77 have separate front-ends but share most of their back-end.

One can go further. As abstraction goes on, the internal representation becomes more language independent, and semantic constructs such as declarations, assignments, calls, IO operations, 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 the procedures. There is an arrow from node A to node B iff A possibly calls B. Recursion leads to cycles. The call graph captures the notions of visibility scope between procedures, that come from modules or classes.
- At the middle level is the control flow graph. There is one flow graph per procedure, i.e. per node in the call graph. The flow graph captures the control flow between atomic instructions. Flow control instructions are represented uniformly inside the control flow graph.
- At the lowest level are abstract syntax trees for the individual atomic instructions. Certain semantic transformations can benefit from the representation of expressions as directed acyclic graphs, sharing common sub-expressions.

To each basic block is associated a symbol table that gives access to properties of variables, constants, function names, type names, and so on. Symbol tables must be nested to implement *lexical 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 [23], [32], [21]. But many analyses are more complex, and are thus 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 [24], 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* [25] 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. The classical solution is to choose a hierarchical model. In this model, information, or at least a computationally expensive part of it, is synthesized. Specifically, it is computed bottom up, starting on the lowest (and smallest) levels of the program representation and then recursively combined at the upper (and larger) levels. Consequently, this synthesized information must be made independent of the context (i.e., the rest of the program). When the synthesized information is built, it is used in a final pass, essentially top down and context dependent, that propagates information from the "extremities" of the program (its beginning or end) to each particular subroutine, basic block, or instruction.

Even then, data flow analyses are limited, because they are static and thus have very little knowledge of actual run-time values. Most of them are *undecidable*; that is, there always exists a particular program for which the result of the analysis is uncertain. This is a stronglimitation, however very theoretical. More concretely, there are always cases where one cannot decide statically that, for example, two variables are equal. This is even more frequent with two pointers or two array accesses. Therefore, in order to obtain safe results, conservative *over-approximations* of the computed information are generated. For instance, such approximations are made when analyzing the activity or the TBR ("To Be Restored") status of some individual element of an array. Static and dynamic *array region analyses* [39], [26] provide very good approximations. Otherwise, we make a coarse approximation such as considering all array cells equivalent.

When studying program *transformations*, one often wants to move instructions around without changing the results of the program. The fundamental tool for this is the *data dependence graph*. This graph defines an order between *run-time* instructions such that if this order is preserved by instructions rescheduling, then the output of the program is not altered. Data dependence graph is the basis for automatic parallelization. It is also useful in AD. *Data dependence analysis* is the static data-flow analysis that builds the data dependence graph.

3.3. Automatic Differentiation and Computational Fluid Dynamics

Participants: Alain Dervieux, Laurent Hascoët, Bruno Koobus.

linearization The mathematical equations of Fluid Dynamics are Partial Derivative Equations, that are discretized and then solved by a computer program. Linearization of these equations, or alternatively linearization of the computer program, gives a modelization of the behavior of the flow 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.

Computational Fluid Dynamics is now able to make reliable simulations of very complex systems. For example it is now possible to simulate completely the 3D air flow around a plane that captures the physical phenomena of shocks and turbulence. The next step in CFD 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 restrict here to gradient descent methods. One risk is obviously to fall into local minima before reaching the global minimum. We do not address this question, although we believe that more robust approaches, such as evolutionary approaches, could benefit from a coupling with gradient descent approaches. Another well-known risk is the presence of discontinuities in the optimized function. We investigate two kinds of methods to cope with discontinuities: we can devise AD algorithms that detect the presence of discontinuities, and we can design optimization algorithms that solve some of these discontinuities.

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

- One can write an *adjoint system*, then discretize it and program it by hand. The adjoint system is a new system, deduced from the original equations, and whose solution, the *adjoint state*, leads to the gradient. A hand-written adjoint is very sound mathematically, because the process starts back from the original equations. This process implies a new separate implementation phase to solve the adjoint system. During this manual phase, mathematical knowledge of the problem can be translated into many hand-coded refinements. But this may take an enormous engineering time. Except for special strategies (see [29]), this approach does not produce an exact gradient of the discrete functional, and this can be a problem if using optimization methods based on descent directions.
- A program that computes the gradient can be built by pure Automatic Differentiation in the reverse mode (*cf* 3.1). It is in fact the adjoint of the discrete functional computed by the software, which is piecewise differentiable. It produces exact derivatives almost everywhere. Theoretical results [28] guarantee convergence of these derivatives when the functional converges. This strategy gives reliable descent directions to the optimization kernel, although the descent step may be tiny, due to discontinuities. Most importantly, AD adjoint is *generated* by a tool. This saves a lot of development and debug time. But this systematic approach leads to massive use of storage, requiring code transformation by hand to reduce memory usage. Mohammadi's work [33] [36] illustrates the advantages and drawbacks of this approach.

The drawback of AD is the amount of storage required. If the model is steady, can we use this important property to reduce this amount of storage needed? Actually this is possible, as shown in [30], where computation of the adjoint state uses the iterated states in the direct order. Alternatively, most researchers [33] use only the fully converged state to compute the adjoint. This is usually implemented by a hand modification of the code generated by AD. But this is delicate and error-prone. The TROPICS team investigate hybrid methods that combine these two extreme approaches.

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.

We will detail some of them in the next sections. These applications require an AD tool that differentiates programs written in classical imperative languages, FORTRAN77, FORTRAN95, C, or C++. We also consider our AD tool TAPENADE as a platform to implement other program analyses and transformations. TAPENADE does the tedious job of building the internal representation of the program and running static data-flow analysis, and then provides an API to build new tools on top of this representation.

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* [34] in weather forecasting or in oceanography. 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. The 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 rapidly boils down to solving an adjoint problem, which can be done though AD [38]. Figure 3 shows an example of a data assimilation exercise using the oceanography code OPA [35] 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.

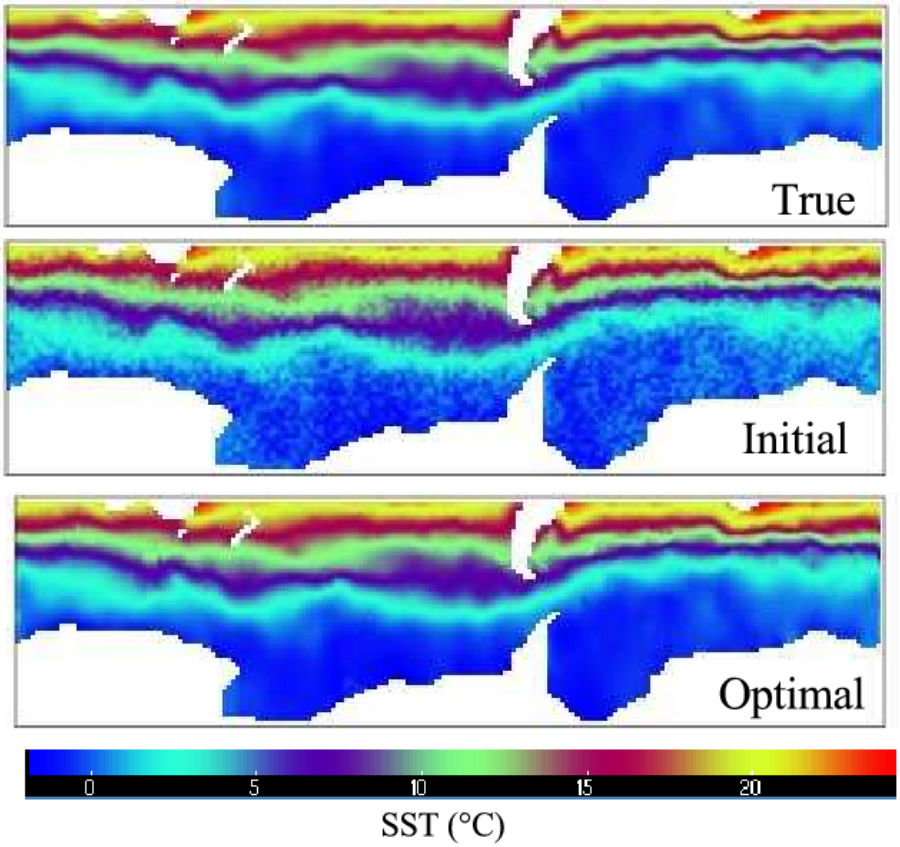

Figure 3. Twin experiment using the adjoint of OPA. We add random noise to a simulation of the ocean state around the Antarctic, and we remove this noise by minimizing the discrepancy with the physical model

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

It has been noticed that 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.

TAPENADE is the Automatic Differentiation tool developed by the TROPICS team. TAPENADE progressively implements the results of our research about models and static analyses for AD. From this standpoint, TAPENADE is a research tool. Our objective is also to promote the use of AD in the scientific computation world, including the industry. Therefore the team constantly maintains TAPENADE to meet the demands of our industrial users. TAPENADE can be simply used as a web server, available at the URL
http://tapenade.inria.fr:8080/tapenade/index.jsp
It can also be downloaded and installed from our FTP server
ftp://ftp-sop.inria.fr/tropics/tapenade/README.html
Documentation is available on our web page
http://www-sop.inria.fr/tropics/
and as an INRIA technical report (RT-0300)
http://hal.inria.fr/inria-00069880

TAPENADE differentiates computer programs according to the model described in section 3.1. It supports three modes of differentiation:

- the *tangent* mode that computes a directional derivative $F'(X).\dot{X}$,
- the *vector tangent* mode that computes $F'(X).\dot{X}_n$ for many directions X_n simultaneously, and can therefore compute Jacobians, and
- the *reverse* mode that computes the gradient $F'^*(X).\overline{Y}$.

The *vector reverse* mode is not implemented, although this could be done if the need arises. Many other modes exist in the other AD tools in the world, that compute for example higher degree derivatives or Taylor expansions. For the time being, we restrict ourselves to first-order derivatives and we put our efforts on the reverse mode. Notice however that higher-order derivatives can be obtained through repeated application of tangent AD on tangent and/or reverse AD.

In addition to classical Type-Checking and Read-Write analysis, TAPENADE performs the following sophisticated static analyses in order to produce an efficient output :

- **Pointer (or Alias) analysis:** For any static program transformation, and in particular differentiation, it is essential to have an accurate knowledge of the possible destinations of each pointer at each code line. Otherwise one must make conservative assumptions that will lead to less efficient code. Our static pointer analysis finds precise information about pointer destinations, taking into account memory allocation and deallocation.
- **Activity:** The end-user has the opportunity to specify which of the output variables must be differentiated (called the dependent variables), and with respect to which of the input variables (called the independent variables). Activity analysis propagates the dependent, backward through the program, to detect all intermediate variables that possibly influence them. Conversely, activity analysis also propagates the independent, forward through the program, to find all intermediate variables that possibly depend on them. Only the intermediate variables that both depend on the independent and influence the dependent are called *active*, and will receive an associated derivative variable. Activity analysis makes the differentiated program smaller and faster.
- **Adjoint Liveness and Adjoint Read-Write:** Programs produced by the reverse mode of AD show a very particular structure, due to the derivative calculations performed in *reverse* order. This has deep consequences on the liveness and Read-Write status of variables, that we can exploit to take away unnecessary instructions and memory usage from the reverse differentiated program. This makes the adjoint program smaller and faster by factors that can go up to 40%.
- **TBR:** The reverse mode of AD, with the Store-All strategy, stores all intermediate variables just before they are overwritten. However this is often unnecessary, because derivatives of some expressions (e.g. linear expressions) only use the derivatives of their arguments and not the original arguments themselves. In other words, the local Jacobian matrix of an instruction may not need all the intermediate variables needed by the original instruction. The *To Be Restored (TBR)* analysis finds which intermediate variables need not be stored during the forward sweep, and therefore makes the differentiated program smaller in memory.

Several other strategies are implemented in TAPENADE to improve the differentiated code. For example, a data-dependence analysis allows TAPENADE to move instructions around safely, gathering instructions to reduce cache misses. Also, long expressions are split in a specific way, to minimize duplicate sub-expressions in the derivative expressions.

The input languages of TAPENADE are FORTRAN77, FORTRAN95, and C. The extension for C has been released in 2008, and is still more experimental than for FORTRAN. Thanks to the language-independent internal representation of programs, as shown on figure 4, this still makes a single and only tool, and every further development benefits to differentiation of each input language.

There are in fact three user interfaces for TAPENADE. One is a simple command that can be called from a shell or from a Makefile. It is recommended for an intensive usage. The second is interactive and graphic, using JAVA SWING components and HTML pages. This second interface allows one to use TAPENADE from WINDOWS as well as LINUX. The third user interface is similar to the second, but runs as a web server. The graphic output interface, shown on figure 5, displays the differentiated programs, with HTML links that implement source-code correspondence, as well as correspondence between error messages and locations in the source.

TAPENADE is now available for LINUX, SUN, MAC-OS, and WINDOWS-XP platforms. TAPENADE is implemented mostly in JAVA, apart from the front-ends which are separated and can be written in their own languages. Several industrial companies have purchased an industrial license for TAPENADE, the software has been downloaded several hundred times, and the web tool served several thousands of true connections (not robots).

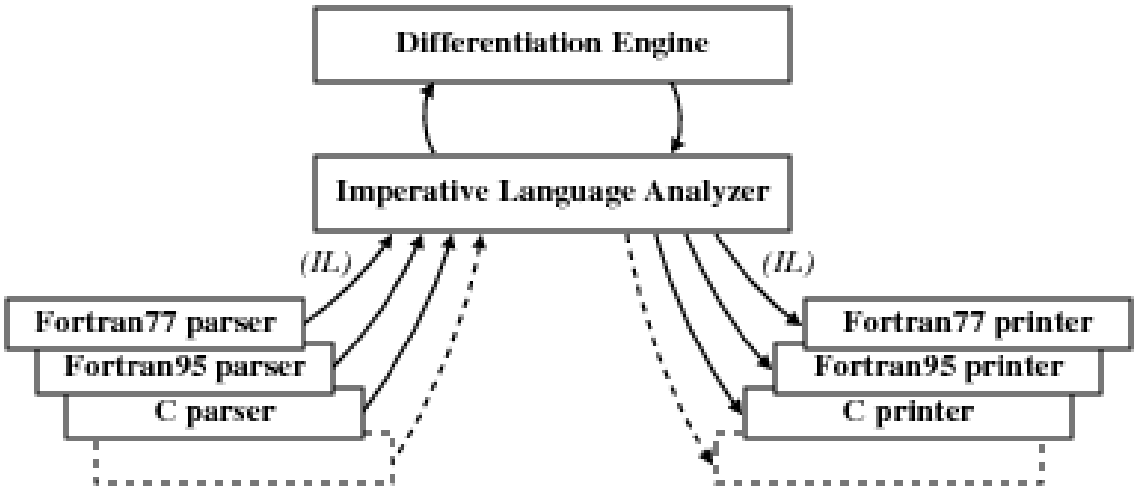

Figure 4. Overall Architecture of TAPENADE

6. New Results

6.1. Automatic Differentiation and parallel codes

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

This study 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.

A static analysis that detects corresponding `isend`'s, `irecv`'s, and `wait`'s is a challenge. It is bound to make conservative assumptions that will degrade the result. Moreover, experiments show that few codes provide clear static information about matching communication routines. MPI tags are often not enough. Therefore we explore a dynamic approach, with a special MPI library enhanced for the reverse mode of AD. The extended MPI communication primitives store at run-time sufficient information for an exact matching of `isend`'s, `irecv`'s, and `wait`'s and their differentiated counterparts. The approach applies equally to AD tools based on program transformation or on operator overloading.

We presented our results [14] at the PDSEC'09 conference this spring, including an experiment on the adjoint of the MIT General Circulation Model. For the record, we also mention an article [37] (forgotten in last year's report) about a proof of correctness of the reverse differentiation scheme for MPI communications.

6.2. TAPENADE for C

Participants: Laurent Hascoët, Valérie Pascual.

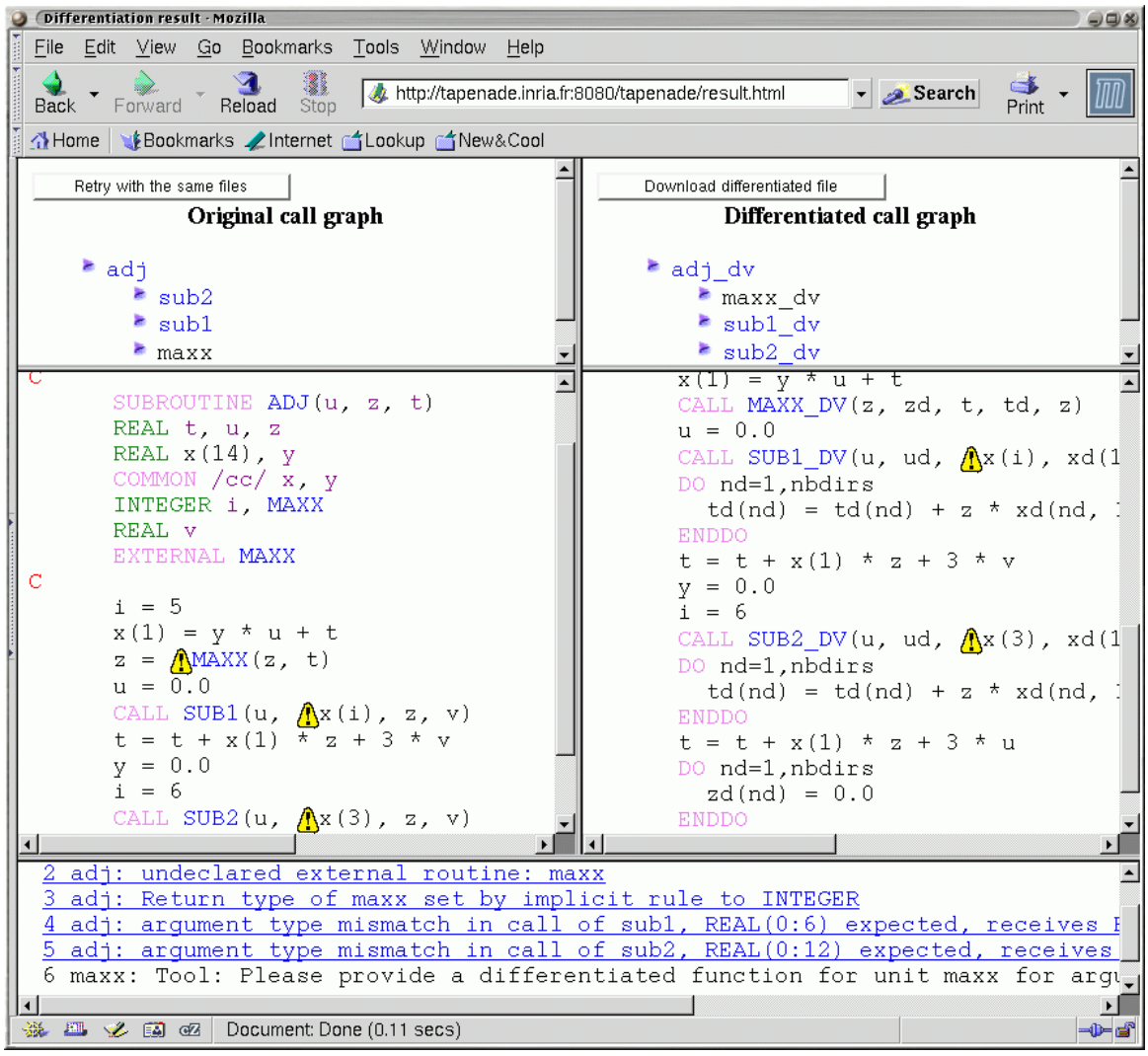

Figure 5. TAPENADE *output interface, with source-code-error correspondence*

The team continued to put considerable effort into Automatic Differentiation of C with TAPENADE. Among other improvements, we underline the following:

- A correct reverse differentiation of arguments that are passed by value instead of by reference.
- An extension of the internal representation of pointers to capture C-style array declarations and indexing.
- An improved treatment of the C `for` loops that are actually equivalent to FORTRAN `do` loops.

6.3. Interface with ADOL-C

Participants: Karim Hossen, Laurent Hascoët.

AD tools come in two main categories.

- Source transformation tools (e.g. TAPENADE), similar to compilers, produce a new differentiated source code from the original. They can perform sophisticated global source analysis and transformation, producing e.g. an efficient, standalone reverse differentiated source program. This remarkable power comes at the cost of a huge tool development effort.
- Overloading-based tools (e.g. ADOL-C) do not change the original source code. They rely on manual modifications of the source code by the user, mostly in the declarations part, that trigger overloading of arithmetic computations with their derivatives. These tools are well suited to local differentiation, like the tangent mode, and all its variants for higher derivatives, Taylor expansions, or interval computations. On the other hand, they are not well suited to the reverse mode.

This year, we developed a prototype interface between the two categories. Namely, we want to use TAPENADE as a front-end for ADOL-C. Overloading-based tools can't perform activity analysis, and therefore cannot tell the user which variables are active and therefore should be turned into the new overloaded type. Thus the user has to do this manually, at the risk of errors. As a front-end for ADOL-C, TAPENADE runs its activity analysis and then automatically changes the declararations of active variables. The new source is then ready to use by ADOL-C without user intervention. TAPENADE can also benefit from this work, as this gives a solution to TAPENADE users who want higher-order derivatives or Taylor series that the team has no time to implement.

Karim Hossen has implemented a new differentiation mode in TAPENADE that just redeclares active variables from `real` or `double` to the ADOL-C defined `adouble` type. Difficulties remain, mainly at the procedure call level, or at places where the activity status changes. The activity status in TAPENADE is dynamic, changing as the program runs. On the other hand, declared types cannot change dynamically, and therefore copies and conversions are sometimes needed for ADOL-C.

Karim Hossen presented his preliminary results to Andrea Walther and her team at the University of Paderborn (Germany), who are now the center of development on ADOL-C.

6.4. Combined Storage and Recomputation for Data-Flow reversal

Participant: Laurent Hascoët.

As explained so many times, the main drawback of the reverse mode of AD is the need to make the temporary variables of the original program available to the derivatives computation, in *reverse order*. This Data-Flow reversal is bound to have a cost in memory space or in duplicate computations. Book chapter [15] provides an in-depth description of the problem.

In the past, we tried to look for a framework to represent both storage and recomputation options, in order to look for optimal combinations. These efforts have not been successful yet.

Despite this lack of a general framework, this year we started to develop 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. The strategy is implemented as an extension of the TBR static data-flow analysis. Experiments with Tapenade show measurable improvements for the Push/Pop memory traffic, and thus for run time.

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, with its optimizations ("Efficient Recomputation Algorithm"). Another goal is to use this implemented strategy as a guideline to again look for a general framework and for the optimum.

6.5. Resolution of linearised systems

Participants: Hubert Alcin, Olivier Allain [Lemma], Anca Belme, Marianna Braza [IMF-Toulouse], Alain Dervieux, Bruno Koobus [Université Montpellier 2], 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 needing exactly each of the iterated solution vector.

In the ECINADS ANR project (starting end of 2009), we plan to design more efficient solution algorithms and to examine the questions risen by their reverse differentiation. The efficiency will be evaluated through the practical scalability on a large number of processors. Scalable efficiency of the reverse differentiated algorithm will be studied. ECINADS associates the university of Montpellier 2, the Institut de Mécanique des fluides de Toulouse and Lemma company.

6.6. Second Derivatives

Participants: Massimiliano Martinelli [UniversitÃ Politecnica delle Marche, Ancona], Alain Dervieux, Laurent Hascoët, Régis Duvigneau [OPALE team].

In the context of the European project NODESIM-CFD, the contribution of Tropics involved 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).

We compared the costs of ToT and ToR in the classical context where the state equation is *implicit*. ToR wins over ToT only when the number n of input parameters is large enough, An earlier result [40] claims that ToT is better for any n. We showed that this earlier result comes from an oversimplification in the evaluation of the algorithm cost. Thanks to the second derivation of CFD kernels, a 3-term Taylor formula gives a second-order reduced order model. The total computational cost of the new reduced model has been compared to other meta models such as Kriging and Radial Basis functions. These results have been presented in a special session on uncertainty management in [11]. As a contribution to NODESIM-CFD, a "Guide for Uncertainty Management in CFD" has been written in collaboration with NUMECA and Vrije Universiteit Brussels [16].

6.7. Correction of approximation errors

Participants: Anca Belme, Alain Dervieux, Massimiliano Martinelli [UniversitÃ Politecnica delle Marche, Ancona].

This subject is becoming an important application of TAPENADE. We investigate the two types of correctors, by direct linearisation and Defect Correction, or by the adjoint-based functional correction. The purpose is to apply these methods to large unsteady flow simulations. These studies contribute to the approximation error section of project NODESIM-CFD. New results have been presented in two NODESIM-CFD seminars in a special session on Uncertainty management [11] and in [13].

6.8. Control of approximation errors

Participants: Frédéric Alauzet [GAMMA team, INRIA-Rocquencourt], Olivier Allain [Lemma], Anca Belme, Alain Dervieux, Damien Guegan [Lemma], Bruno Koobus, 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 by 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. 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 [12].

Together with project-team GAMMA and PUMAS, TROPICS contributes this research on mesh adaptation methods in aeronautics to the HISAC IP European project.

7. Dissemination

7.1. Links with Industry, Contracts

- Several industrial companies have purchased an industrial license for TAPENADE. Rolls-Royce UK had a licence that expired in 2009, and renewed it for 5 years. TAPENADE is also used by many academic institutions for education and research. Many users cannot be identified, because the log files of our web and ftp servers give little information. However, we are aware of TAPENADE regular use by researchers in Argonne National Lab. (Illinois, USA), the Federal Reserve (Washington DC, USA), CSIRO Hobart (Australia), NAL Bangalore (India), Cranfield university (UK), Oxford university (UK), Queen Mary university London (UK), RWTH Aachen (Germany), Humboldt university Berlin (Germany), German Aerospace Center (Germany), DLR (Germany), General Electric Deutschland (Germany), University of Bergen (Norway), ISMAR-CNR Venezzia (Italy), Alenia (Italy), Dassault Aviation (France), INSA Toulouse (France), Université Montpellier 2 (France), CMAP Ecole Polytechnique (France) ...

  Here are some recent statistics on the use of TAPENADE: There are roughly 2 releases per year. The latest release has been downloaded 123 times from our ftp server, between July 2008 and January 2009 from 23 different countries including Germany (29), France (23), USA (11), Italy (6). The TAPENADE web server has been used more than 5000 times since its creation in 2002 (Actual uses only, not robots). The current rate is 800 uses (sessions) per year. These uses come from about 260 different geographical locations, from 42 different countries including France (48), Germany (33), USA (21), UK (17). More than 100 users gave us their name and application when using the Tapenade web server, and 74 have registered in the “tapenade-users” mailing list.
- TROPICS participates in the European IP project HISAC, ending in 2009, driven by Dassault Aviation and involving 31 partners. TAPENADE has been made available to partners. TROPICS, GAMMA, and SMASH designed mesh adaptation methods for evaluating the sonic boom and a combined mesh-adaptative/shape optimisation method for reducing the sonic boom.

- 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).
- TROPICS participates in the project LEFE, “Les Enveloppes Fluides et l’Environnement”, which is a CNRS API project accepted in 2007. Our contribution is to provide the automatic production of the adjoint of OPA [35] (ORCA-2 configuration), with the help of TAPENADE.
- TROPICS participates in the European STREP project NODESIM (Non-Deterministic Simulation for CFD-based design methodologies), driven by Numeca (Belgium). TROPICS and OPALE contribute to application of AD to build reduced models using first and second derivatives. We design robust optimization strategies, and correctors for approximation errors.
- TROPICS 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.

7.2. Conferences and workshops

- TROPICS successfully went through the complete INRIA team evaluation process. Evaluation meeting took place on march 17-18.
- Alain Dervieux presented the team’s results on AD for uncertainties and error correction at the 44th AAAF congress in Nantes, march 23-25, and at the MAMERN conference in Pau, june 8-11.
- Anca Belme gives lectures to 3rd year students at tthe university of Nice.
- Anca Belme represented TROPICS at the 5th NODESIM-CFD meeting in Trieste, may 25-26.
- Anca Belme presented new results on "Correction d’erreurs numériques par linéarisés et adjoints" as a poster at SMAI 2009, la Colle sur Loup, may 25-29.
- Laurent Hascoët presented the team’s results on reverse AD of MPI-parallel codes at the PDSEC’09 conference in Rome, Italy, may 25-29.
- Laurent Hascoët, Jean Utke, and Uwe Naumann met in Aachen Germany for 10 days, and on this occasion presentations were organized for students.
- Laurent Hascoët presented Karim Hossen’s results at the 8th European AD Workshop hosted by NAG in Oxford, UK, july 13-14.
- Laurent Hascoët attended the Eva-Flo meeting in Lyon, september 22-23.
- Anca Belme and Alain Dervieux presented the contribution of TROPICS to the 6th NODESIM-CFD meeting in Brussels, october 29-30.
- Laurent Hascoët presented TAPENADE and organized a hands-on session at the ERCOFTAC school on optimization at Humboldt University Berlin, november 11-13.
- Laurent Hascoët is on the organizing commitee of the European Workshops on Automatic Differentiation. The team organized the 9th edition at INRIA Sophia-Antipolis, november 26-27.
- Laurent Hascoët is on the program committee for the 1st workshop on Automated Program Generation for Computational Science at ICCS 2010, Amsterdam, May 31 - June 2, 2010.
- Laurent Hascoët was on the PhD jury for Thomas Migliore (university of Nice).
- Alain Dervieux was on the PhD jury for Yogesh Parte (university of Toulouse) and Hilde Ouvrard (university Montpellier 2). university of 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", vol. 20, n^{o} 7, 2006, 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", vol. 18, n^{o} 5, 2003, 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, chap. 35, Springer, 2001, p. 299-304.

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

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

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

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

Year Publications

Articles in International Peer-Reviewed Journal

[11] A. BELME, M. MARTINELLI, L. HASCOËT, V. PASCUAL, A. DERVIEUX. *AD-based perturbation methods for uncertainties and errors*, in "International Journal of Engineering System Modelling and Simulation", 2009, special issue after 44th AAAF, march 2009, Nantes, France IT .

International Peer-Reviewed Conference/Proceedings

[12] F. ALAUZET, A. DERVIEUX, A. LOSEILLE. *Fully anisotropic goal-oriented mesh adaptation for the Euler equations*, in "Proceedings of ADMOS2009", 2009.

[13] A. BELME. *Correction d'erreurs numériques par linéarisés et adjoints*, in "poster presented at SMAI 2009 - La Colle sur Loup", 2009.

[14] J. UTKE, L. HASCOËT, P. HEIMBACH, C. HILL, P. HOVLAND, U. NAUMANN. *Toward Adjoinable MPI*, in "Proceedings of the 10th IEEE International Workshop on Parallel and Distributed Scientific and Engineering, PDSEC'09", 2009 US DE .

Scientific Books (or Scientific Book chapters)

[15] L. HASCOËT. *Reversal Strategies for Adjoint Algorithms*, in "From Semantics to Computer Science. Essays in memory of Gilles Kahn", Cambridge University Press, 2009, p. 487–503.

Research Reports

[16] A. BELME, C. DINESCU, A. DERVIEUX, R. DUVIGNEAU, L. HASCOËT, C. HIRSCH, C. LACOR, M. MARTINELLI, V. PASCUAL, S. SMIRNOV. *Guide for uncertainties and error management in CFD*, n^{o} D4.1-07, NODESIM, 2009, deliverable BE IT .

[17] A. BELME, H. OUVRARD. *Combining a Mass Matrix formulation and a high order dissipation for the discretisation of turbulent flows*, n^{o} 7079, INRIA, 2009, http://hal.inria.fr/inria-00429062/en/, Research Report.

[18] A.-C. LESAGE, A. DERVIEUX. *Conservation correction by dual Level Set*, INRIA, 2009, http://hal.inria.fr/inria-00430191/en/, Research Report.

[19] H. OUVRARD, B. KOOBUS, M.-V. SALVETTI, S. CAMARRI, S. WORNOM, A. DERVIEUX. *Parallel Simulation of Complex Unsteady Flows with Variational Multiscale LES and Hybrid RANS/LES*, n^{o} 6917, INRIA, 2009, http://hal.inria.fr/inria-00381570/en/, Research ReportIT.

References in notes

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

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

[22] A. CARLE, M. FAGAN. *ADIFOR 3.0 overview*, n^{o} CAAM-TR-00-02, Rice University, 2000, Technical report.

[23] 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, http://hal.inria.fr/inria-00076140, Also, Information Processing Society of Japan, Technical Memorandum PL-86-6. Also INRIA research report # 416.

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

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

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

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

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

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

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

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

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

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

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

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

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

[37] U. NAUMANN, L. HASCOËT, C. HILL, P. HOVLAND, J. RIEHME, J. UTKE. *A Framework for Proving Correctness of Adjoint Message-Passing Programs*, in "Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface", Springer-Verlag, Berlin, Heidelberg, 2008, p. 316–321.

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

[39] 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.

[40] A. C. TAYLOR III, L. L. GREEN, P. A. NEWMAN, M. M. PUTKO. *Some advanced concepts in discrete aerodynamic sensitivity analysis*, in "AIAA Journal", vol. 41, n° 7, 2003, p. 1224–1229.