INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team tropics

Transformations et Outils Informatiques pour le Calcul Scientifique

Sophia Antipolis

THEME NUM

Table of contents

1. Team

Head of project team
Laurent Hascoët [CR INRIA]

Vice-head of project team
Valérie Pascual [CR INRIA]

Administrative assistant
Claire Senica [TR INRIA]

Staff members
Alain Dervieux [DR INRIA]

Ph. D. students
Mauricio Araya-Polo
Benjamin Dauvergne

Junior technical staff
Christophe Massol

Research Engineer
Janet Bertot

Partner scientists
Bruno Koobus [Université de Montpellier 2]
Mariano Vázquez [Universitat de Girona, Spain]

2. Overall Objectives

2.1. Overall Objectives

The TROPICS team is at the junction of two research domains:

- **AD:** On one hand, we study software engineering techniques, to analyze and transform programs semi-automatically. In the past, we developed semi-automatic parallelization strategies aiming at SPMD parallelization. Presently, we focus on Automatic Differentiation (AD). AD transforms a program `P` that computes a function F, into a program `P'` that computes some derivatives of F, analytically. In particular, the so-called *reverse mode* of AD yields gradients. However, this reverse mode remains very delicate to use, and its implementation requires carefully crafted algorithms.
- **CFD application of AD:** On the other hand, we study the application of AD, and particularly of the adjoint method, to Computational Fluid Dynamics. This involves necessary adaptation of optimization strategies. This work applies to two real-life problems, optimal shape design and mesh adaptation.

The second aspect of our work (optimization in Scientific Computing), is thus at the same time the motivation and the application domain of the first aspect (program analysis and transformation, and computation of gradients through AD). Concerning AD, our goal is to automatically produce derivative programs 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 project. Apart from being an AD tool, TAPENADE is also a platform for other analyses and transformations of scientific programs. TAPENADE is easily available. We provide a web server, and alternatively a version can be downloaded from our web server. Practical details can be found in section 5.1.

Our present research directions are :

- Modern numerical methods for finite elements or finite differences: multigrid methods, mesh adaptation.
- Optimal shape design, in the context of fluid dynamics: for example shape optimization of the wings of a supersonic aircraft, to reduce sonic bang. Also, new optimization tactics combining interior point, SQP or one-shot algorithms.
- Automatic Differentiation : reduce runtime and memory consumption when computing gradients ("adjoints") or Jacobian matrices, differentiate parallel programs, differentiate particular algorithms in a specially adapted manner, validate the derivatives.
- Common tools for program analysis and transformation: adequate internal representation, Call Graphs, Flow Graphs, Data-Dependence Graphs.

3. Scientific Foundations

3.1. Automatic Differentiation

Keywords: *adjoint models*, *automatic differentiation*, *optimization*, *program transformation*, *scientific computing*, *simulation*.

Participants: Mauricio Araya-Polo, Benjamin Dauvergne, Laurent Hascoët, Christophe Massol, 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, that returns 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 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, some 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 [41], adjoint problems [36], 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 [38]. 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.

We can 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 $f'^*_{k+1}(X_k)$. There are two strategies for that:

- **Recompute All (RA):** the X_k is recomputed when needed, restarting P on input X_0 until instruction I_k. The TAF [34] 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 [29] and TAPENADE tools use this strategy. Brute-force SA strategy has a linear memory cost with respect to p.

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

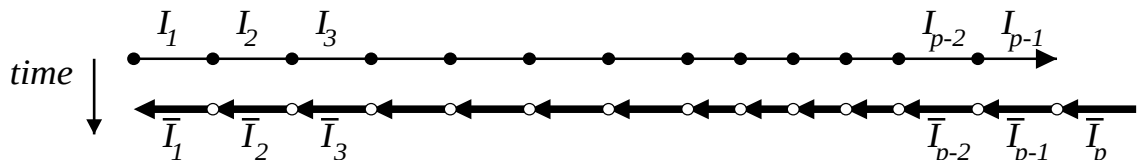

Figure 1. The "Store-All" tactic

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 **C** of the program is illustrated on figure 2. During the forward sweep, no value is stored while in **C**. Later, when the backward sweep needs values from **C**, 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 **C**. This snapshot is shown on figure 2 by slightly bigger black and white dots.

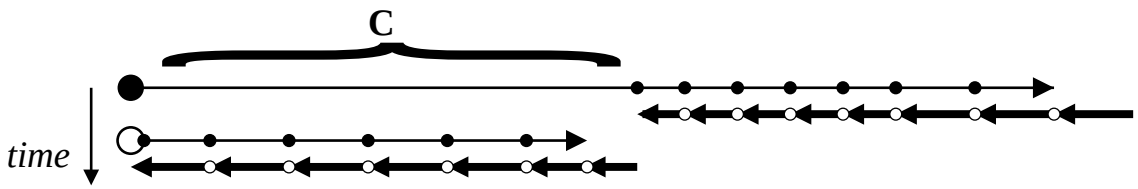

*Figure 2. Checkpointing **C** with the "Store-All" tactic*

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

3.2. Static Analyses and Transformation of programs

Keywords: *abstract interpretation, abstract syntax tree, compilation, control flow graph, data dependence graph, data flow analysis, program transformation, static analysis.*

Participants: Mauricio Araya-Polo, Benjamin Dauvergne, Laurent Hascoët, Christophe Massol, 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 analyses 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 analyses are 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 analyses [27] 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 a 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. Analyses 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 analyses can be defined on this internal representation, which is largely language independent. The simplest analyses on trees can be specified with inference rules [30], [39], [28]. 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 [31], 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* [32] 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 strong, yet very theoretical limitation. 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* [44], [33] 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

Keywords: *adjoint methods, adjoint state, computational fluid dynamics, gradient, linearization, optimization*.

Participants: Alain Dervieux, Laurent Hascoët, Bruno Koobus, Mariano Vázquez.

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, like in acoustics, or when one wants the sensitivity of the system with respect to one parameter, like 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 [36]), 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 [35] 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 [40] [42] 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 [37], where computation of the adjoint state uses the iterated states in the direct order. Alternatively, most researchers (see for example [40]) 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 data assimilation or parameter estimation,
- 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 then provides an API to build new tools on top of this representation. One application of TAPENADE is therefore to build prototypes of new program analyses.

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

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. Another example is *data assimilation* in weather forecasting. The initial state of the simulation conditions the quality of the weather 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 [43].

4.4. Linearization

To simulate 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 parameter. 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 [contact], Mauricio Araya-Polo, Benjamin Dauvergne, Christophe Massol, 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, and therefore in 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

A documentation is available on our web page
http://www-sop.inria.fr/tropics/
and as an INRIA technical report (RT-0300)
http://www.inria.fr/rrrt/rt-0300.html

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 $F'^*(X).\overline{Y}$.

A obvious fourth mode could be the *vector reverse* mode, which is not yet implemented. 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. But as we said before, we also view TAPENADE as a platform to build new program transformations, in particular new differentiations.

Like any program transformation tool, TAPENADE needs sophisticated static analyses in order to produce an efficient output. Concerning AD, the following analyses are a must, and TAPENADE now performs them all:

- **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 the dependent. Conversely, activity analysis also propagates the independent, forward through the program, to find all intermediate variables that possibly depend on the independent. 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.
- **Read-Write:** Each procedure has a number of arguments, that may be inputs, outputs, or both. Compilers use this to remove useless arguments. TAPENADE uses Read-Write information more specifically to detect aliasing, which is very harmful in AD, and to reduce the size of snapshots needed by checkpointing in the reverse mode. Read-Write analysis makes the differentiated program safer and less costly in memory space.
- **Adjoint Liveness and Read-Write:** Programs produced by the reverse mode of AD show a very particular structure, due to their mechanism to restore intermediate values of the original program in the *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 (adjoint) 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 today are FORTRAN77 and FORTRAN95. Notice however that the internal representation of programs is language-independent, as shown on figure 5, so that extension to other languages should be easier.

There are two user interfaces for TAPENADE. One is a simple command that can be called from a shell or from a Makefile. The other is interactive, using JAVA SWING components and HTML pages. This interface allows one to use TAPENADE from WINDOWS as well as LINUX. The input interface, shown on figure 3, lets one specify interactively the routine to differentiate, its independent inputs and dependent outputs.

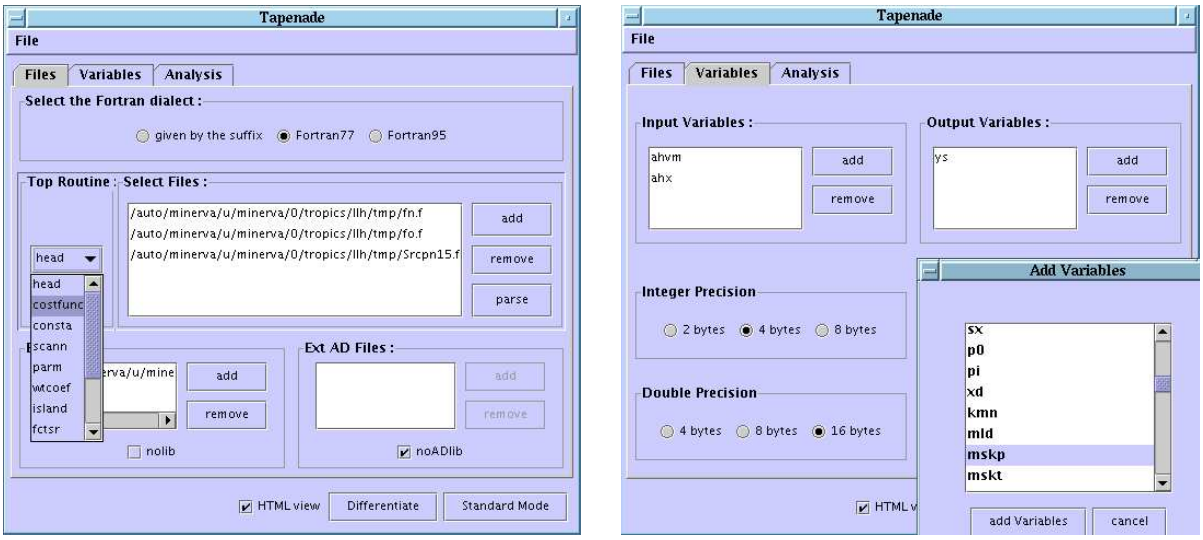

Figure 3. TAPENADE *two successive query windows from the input interface*

The output interface, shown on figure 4, 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, or WINDOWS-XP platforms.

Figure 5 shows the architecture of TAPENADE. It is implemented mostly in JAVA, apart from the front-ends which are separated and can be written in their own languages.

Notice the clear separation between the general-purpose program analyses, based on a general representation, and the differentiation engine itself. Other tools can be built on top of the Imperative Language Analyzer platform.

The end-user can also specify properties of external or black-box routines. This is essential for real industrial applications that use many libraries. The source of these libraries is generally hidden. However AD needs some information about these black-box routines in order to produce efficient code. TAPENADE lets the user specify this information in a separate signature file.

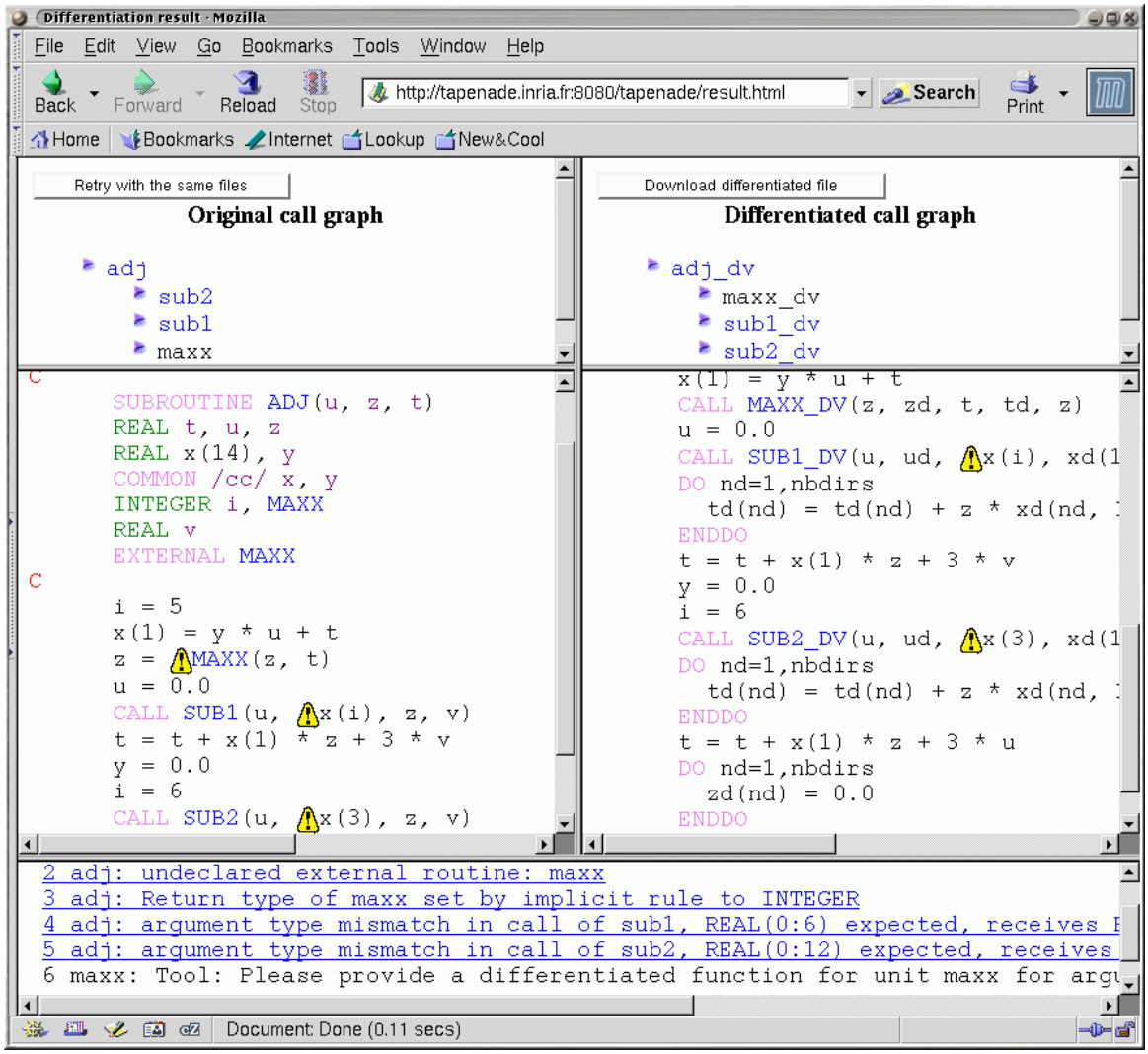

Figure 4. TAPENADE *output interface, with source-code-error correspondence*

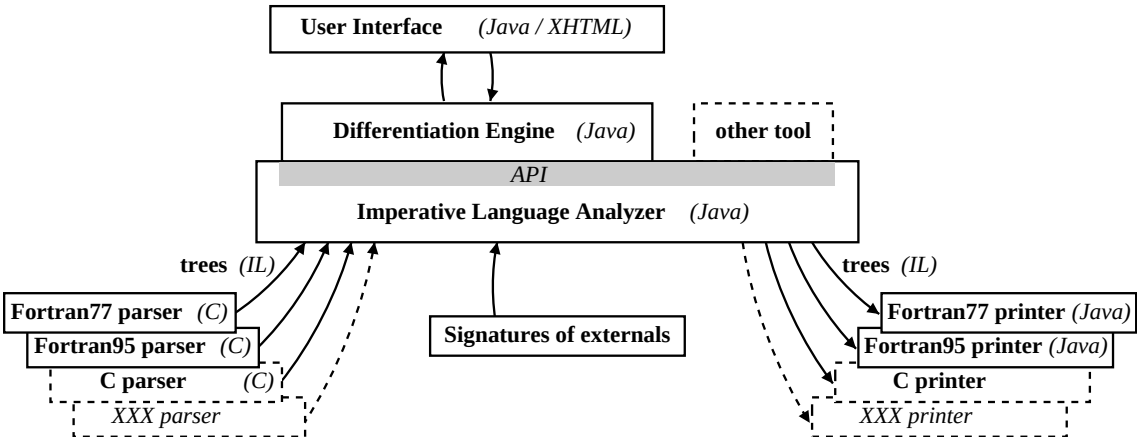

Figure 5. Overall Architecture of TAPENADE

6. New Results

6.1. Data-flow properties of adjoint programs

Keywords: *checkpointing, data-flow analyses, reverse mode of AD, snapshots, static analyses.*

Participants: Mauricio Araya-Polo, Benjamin Dauvergne, Laurent Hascoët.

This year we focused on the properties of checkpointing in reverse differentiated programs. Last year, we have given the data-flow equations of the essential analyses for reverse AD, namely activity analysis, adjoint-liveness analysis, TBR analysis, and adjoint-write analysis. These equations were derived from classical data-flow analysis, which gave a proof of their correctness, in the case where no checkpointing was used. This year we studied the more complex case of checkpoints.

Checkpointing requires to save a subset of the variables (the "snapshot") at a given instant in execution. This snapshot interferes with the variables saved by the TBR analysis, so that many choices are available. Among those, many are "optimal" choices, but none can be an absolute "optimum" for all cases.

Starting back from classical data-flow theory, we derived the equations of all possible optimal choices. We also studied the efficiency of a number of these choices on real applications taken from our validation tests suite. The solution which gives the best results is now implemented as the default strategy in TAPENADE.

We presented this question to the other AD researchers during the first European AD workshop, which we organized in Nice in April. We presented our results during the 2nd workshop of this series, in Cranfield UK in November. An article is submitted to the ICCS 2006 conference in Reading UK.

Still on checkpointing, Mauricio Araya has been developing the so-called "split-mode" of reverse AD. It amounts to modifying the structure of differentiated programs, to give the end-user the choice *not* to checkpoint some procedure calls of the original program. This new flexibility can dramatically improve efficiency of the differentiated programs, especially for calls to small subroutines. This is strongly related to the PhD subject of Benjamin Dauvergne, who explores strategies to choose which procedure calls must be checkpointed or not, based on external information such as execution profiles.

In parallel, Benjamin Dauvergne started to study selective introduction of recomputation to reduce the cost due to storage of intermediate values in reverse AD. Based on properties of data-dependencies, we are able to find which values can actually be recomputed, and to compare the cost of recomputation with the cost of simple storage. Recomputation is not limited to computed values: it also applies to the discrete values involved

in the flow of control, e.g. the results of tests in conditionals. Here also, recomputation can be of great interest by reducing the memory storage due to control.

6.2. Automatic estimation of the validity domain of derivatives

Keywords: *discontinuities, validity of derivatives.*

Participants: Mauricio Araya-Polo, Laurent Hascoët.

Automatic Differentiation gives analytical derivatives, e.g. gradients, based on local properties of the function at the given input point. These derivatives are then used to extrapolate the behavior of the function around this point. Since the analysis is local, these derivatives do not take into account discontinuities that may occur in the neighborhood of the current point. These discontinuities basically correspond to conditionals in the source program.

At the end of last year research, we proposed an algorithm to evaluate the size of the neighborhood around the current point, in which the function remains differentiable. This algorithm is very expensive. We proposed and implemented a limited algorithm that evaluates this size only in one dimension, which could be the descent direction in gradient-based optimization.

This year we made further experiments on larger applications. Mauricio Araya presented his results at the French-Latin American congress on Applied Mathematics in Santiago, Chile [20]. He made a deeper presentation at the WSEAS'05 conference in Cancún, Mexico. We are pleased to highlight that he won the best student paper award. His presentation was then made an article in the "WSEAS transactions on circuits and systems" journal [12].

Mauricio Araya's algorithm is now implemented as a standard differentiation mode in TAPENADE version 2.1.

6.3. Extensions and new functionalities in tapenade

Keywords: *Automatic Differentiation, Fortran90, Tapenade, data-flow analysis, pointer analysis, static analysis.*

Participants: Mauricio Araya-Polo, Benjamin Dauvergne, Laurent Hascoët, Christophe Massol, Valérie Pascual.

This year's development followed mostly two directions: Fortran90 and pointer analysis.

Regarding Fortran90, Valérie Pascual continued extension of differentiation to additional features of Fortran90, which were required by the large applications we have been working on (*cf* 6.4). Let's quote only a few:

- global variables in modules, especially when they have the `SAVE` attribute.
- selective use of only some of the functions and variables defined by a module, combined with renaming
- overloading of standard operations, including assignment.
- array statements that use any combination of `WHERE`, `SUM`, and `MASK` constructs.

Regarding pointer analysis, Christophe Massol developed a first version of the pointer analysis module. It takes into account memory allocation, interprocedural use of pointers, and recursive nested data structures with pointers. We collaborate with an end user from the climate group at Argonne National Laboratory, who has received an alpha version of the pointer analysis module and has accepted to test it on a model for the simulation of sea ice.

Extension of Automatic Differentiation (tangent mode) to programs with pointers will start soon. Adaptation of the model of reverse mode AD is far more complex and is still an open research problem, which we will study in the future.

6.4. Differentiation of large real applications

Keywords: *Agronomy*, *Automatic Differentiation*, *Oceanography*, *Tapenade*, *Variational Data Assimilation*.

Participants: Bruno Ferron [IFREMER, Brest], Laurent Hascoët, Claire Lauvernet [INRA, Avignon].

We studied application of Automatic Differentiation to several very large scientific computation codes.

Bruno Ferron of IFREMER Brest gave us the latest Fortran90 version of the ocean simulation code OPA, version 9.0, developed mainly by LODYC lab. at Paris 6 university. This very large application (80,000 lines) gave us a fair amount of debugging work in the Fortran90 version of TAPENADE, but we eventually obtained a validated adjoint code. It is now possible to obtain an adjoint code for any further version of OPA in a matter of minutes. Performances are satisfying, exhibiting an impressive slowdown factor of *only* seven between the original code and its AD adjoint, on a test-case reduced in size. Work is in progress to extend this experiment to a realistic larger test-case.

Claire Lauvernet at INRA Avignon gave us two codes devoted to simulation of the images seen by a satellite as a function of the nature and state of the soil and the plant cover. This code will be used for data assimilation of the parameters of the actual vegetation from the observed satellite images. Classically, this requires an adjoint model which can be created by Automatic Differentiation in reverse mode of the simulation code. One of these codes is called STICS and its differentiation was a large part of Claire Lauvernet's PhD thesis, defended in April. The other code is called MULTISAIL. Its principal interest is that it couples two simulation codes, therefore bringing an extra level of complexity. We were able to create the adjoint code automatically with TAPENADE, and the results were presented in the ISPMSRS conference in Beijing, China [24]. After the conference, this work was selected for publication of an extended version in the "Remote Sensing of Environment" journal.

6.5. Optimal control

Keywords: *adjoint model*, *gradient*, *optimal control*, *optimum design*.

Participants: Francois Courty, Bruno Koobus, Alain Dervieux, Laurent Hascoët, Mariano Vázquez, Bijan Mohammadi [Université de Montpellier].

Now that simulation is well mastered by research groups in industry, optimization is naturally the next frontier. Optimization problems in aerodynamics are still very difficult, because they require a enormous computing power. To meet these needs, our investigations in AD are focused on the reverse mode, which is an elegant way to obtain the adjoints that optimization uses.

The reverse mode, and the subsequent adjoint state, are the best way to get the gradients when the number of parameters is large. This corresponds to what happens in industry. For example, the optimization of a dozen of scalar parameters will not produce an optimal shape for an aircraft, because an accurate description of a shape requires hundreds of parameters. Even when shapes are described by mathematical functions describing surfaces or volumes, the number of scalar parameters has to be relatively large to obtain an accurate description.

We also work on the optimization of meshes using adjoints built with AD tools. This research is made within an European project HISAC on supersonic aircraft.

These large scale optimization problems share the common requirement for adjoint systems that are efficient with respect to the size of the parameter space, and with respect to the length and complexity of the original simulation. They also require that we use efficient preconditionners for the solution steps involved. This problematic is presented in greater detail in our paper accepted to the post-SAROD workshop in Bangalore, India [22].

6.6. Multidisciplinary optimization

Keywords: *fluid*, *gradient*, *optimization*, *structure*.

Participants: Alain Dervieux, Bruno Koobus, Mariano Vázquez, Charbel Farhat [Stanford University], Francois Beux [Scuola Normale Superiore di Pisa].

The optimization of a complex product like an airplane needs to be done by taking into account simultaneously as many different physical effects as possible. Practically, it is useless to search the optimal aerodynamic shape of an aircraft if the influence on the structure is not anticipated. Many of the physics to take into account are described by complex and computer intensive models. We proposed a new coupling algorithm for the aerodynamic shape optimization of an aircraft geometry deformed by the wind [26], [19]. Francois Beux and Alain Dervieux are co-advisors of the thesis of Massimiliano Martinelli, on the subject of Multidisciplinary shape optimization.

6.7. Mesh adaptation

Keywords: *adjoint, mesh adaptation, optimization.*

Participants: Frederic Alauzet [GAMMA], Alain Dervieux, Francois Courty, Bruno Koobus, Adrien Loseille, Youssef Mesri, Tristan Roy.

We continued our investigations on the use of smart mesh adaptation for increasing the convergence order of a series of adapted computations. For optimal control and optimum design (*cf* 6.5) we developed innovative strategies to solve the adjoint problem. These strategies can also be applied to a different context, namely mesh adaptation. This is possible if we can map the mesh adaptation problem into a differentiable optimal control problem. To this end, we introduced a new methodology that consists in setting the mesh adaptation problem under the form of a purely functional one: the mesh is reduced to a continuous property of the computational domain, the continuous metric, and we minimize a continuous model of the error resulting from that continuous property. Then the problem of searching an adapted mesh is transformed in the research of an optimal metric.

In the case of mesh interpolation minimization, the optimum is given by a closed formula and gives access to a complete theory demonstrating that second order accuracy can be obtained on discontinuous field approximation [14].

In the case of adaptation for Partial Differential Equations, we obtain an Optimal Control problem. It involves a state equation and the optimality is expressed in terms of an adjoint state, which can be obtained using AD. In our first prototypes, the one-shot-SQP algorithm has been applied successfully [26]. This is the focus of a cooperation with project-team GAMMA at INRIA-Rocquencourt (Paul-Louis George, Frédéric Alauzet, Adrien Loseille) for the INRIA contribution to the HISAC IP European project in association with 30 other partners in aeronautics.

We present our results in detail in our paper accepted to the SAROD conference in Hyderabad, India [23].

7. Dissemination

7.1. Links with Industry, Contracts

- Alain Dervieux received a prize (prix Marcel Dassault) from the French Academy of Sciences in august.
- Two industrial users purchased a commercial license for TAPENADE: Rolls-Royce (aeronautic engines) and Cargill (agronomy industry).
- The National Aerospace Laboratory in Bangalore, India asked for a free research license for TAPENADE. The license was signed at the end of the year.
- TROPICS is leader of a project “Optimisation de forme et adaptation de maillage pour le bang supersonique” supported by the Comité d’Orientation Supersonique of the French ministry of Research. Our partners are the université de Montpellier and the Gamma project in Rocquencourt.
- TROPICS participates in the European project HISAC, accepted last year and truly starting at the end of this year.

- TROPICS participates in two other requests for European funding (1 STREPS and 1 Marie-Curie), whose results will be known by the end of the year.
- We are aware of TAPENADE regular use by research groups in Argonne National Lab. (USA), Cranfield university (UK), Oxford university (UK), RWTH Aachen (Germany), and Humboldt university Berlin (Germany).

7.2. Conferences and workshops

- Alain Dervieux made several presentations after receiving the “prix Marcel Dassault” of the Academy of Sciences.
- Alain Dervieux presented his work on mesh adaptation at Oxford university and at université de Montréal.
- Laurent Hascoët defended his “habilitation” thesis in January in Sophia-Antipolis, on “Static Analyses and Transformations of Programs: from Parallelization to Differentiation”.
- The team organized the 1st edition of the European AD workshop in Nice from the 11th to the 15th of April. This follows the former series of UK AD workshops in Hatfield and Cranfield. Benjamin Dauvergne made a presentation on checkpointing.
- Mauricio Araya gave two talks on the problem of domain of validity of derivatives at the 7th French-Latin American Congress on Applied Mathematics, in Santiago, Chile, and at the WSEAS’05 conference in Cancun, Mexico. This second talk won the best student paper award.
- Mauricio Araya also presented his research at an internal “séminaire croisé” at INRIA Sophia-Antipolis, and to visiting executives of Chilean CONICYT.
- Laurent Hascoët made two lectures on Automatic Differentiation for Optimum Design, one at the ERCOFTAC "Introductory Course to Design Optimization", April 6-8, in Manchester, UK, and one at the CEA-EDF-INRIA “école d’été” on numerical analysis, June 27th till July 8th in Paris.
- Laurent Hascoët made a presentation on checkpoints at the 2nd European AD workshop in Cranfield, UK in November.
- Laurent Hascoët served as president of the PhD jury of Claire Lauvernet, of INRA Avignon, France, who used Automatic Differentiation for agricultural data assimilation from satellite observations.

8. Bibliography

Major publications by the team in recent years

[1] F. COURTY. *Optimisation Différentiable en Mécanique des Fluides Numérique*, Ph. D. Thesis, Université Paris-sud, 2003.

[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° 5, 2003, p. 615-627.

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

[4] 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.

[5] L. HASCOËT. *A method for automatic placement of communications in SPMD parallelisation*, in "Parallel Computing Journal", n° 27, 2001, p. 1655-1664.

[6] L. HASCOËT, V. PASCUAL. *TAPENADE 2.1 user's guide*, Technical report, n° 300, INRIA, 2004, http://www.inria.fr/rrrt/rt-0300.html.

[7] L. HASCOËT. *The Data-Dependence Graph of Adjoint Programs*, Research Report, n° 4167, INRIA, 2001, http://www.inria.fr/rrrt/rr-4167.html.

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

[9] L. HASCOËT, M. VÁZQUEZ, A. DERVIEUX. *Automatic Differentiation for Optimum Design, applied to Sonic Boom reduction*, in "Proceedings of the International Conference on Computational Science and its Applications, ICCSA'03, Montreal, Canada", V. KUMAR, ET AL. (editors). , LNCS 2668, Springer, 2003, p. 85-94.

[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.

Doctoral dissertations and Habilitation theses

[11] L. HASCOËT. *Analyses statiques et transformations de programmes: de la parallélisation à la différentiation*, Habilitation, Université de Nice Sophia-Antipolis, 2005.

Articles in refereed journals and book chapters

[12] M. ARAYA-POLO, L. HASCOËT. *Certification of Directional Derivatives Computed by Automatic Differentia-*

tion, V. MLADENOV (editor). , WSEAS TRANSACTIONS on CIRCUITS and SYSTEMS, WSEAS, Athens, Greece, 2005.

[13] F. COURTY, A. DERVIEUX. *A SQP-like one-shot algorithm for optimal shape design*, V. SELMIN (editor). , to appear, Springer, Notes on Numerical Fluid Dynamics, 2005.

[14] F. COURTY, D. LESERVOISIER, P.-L. GEORGE, A. DERVIEUX. *Continuous metrics and mesh optimization*, in "Applied Numerical Mathematics", to appear, 2005.

[15] 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.

[16] L. HASCOËT, R.-M. GREBORIO, V. PASCUAL. *Computing Adjoints by Automatic Differentiation with TAPENADE*, B. SPORTISSE, F.-X. LEDIMET (editors). , to appear, Springer, 2005.

[17] L. HASCOËT, V. PASCUAL, D. DERVIEUX. *Automatic Differentiation with TAPENADE*, V. SELMIN (editor). , to appear, Springer, Notes on Numerical Fluid Dynamics, 2005.

[18] V. PASCUAL, L. HASCOËT. *Extension of TAPENADE towards Fortran 95*, 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.

[19] M. VÁZQUEZ, A. DERVIEUX, B. KOOBUS. *A methodology for the shape optimization of flexible wings*, in "Eng. Comp.", 2005.

Publications in Conferences and Workshops

[20] M. ARAYA-POLO. *On validity of derivatives computed by Automatic Differentiation*, in "VII French-Latin American Congress on Applied Mathematics, January 14-18, Santiago, Chile", 2005.

[21] B. COURTY, T. ROY, B. KOOBUS, M. VÁZQUEZ, A. DERVIEUX. *Error analysis for P1-exact schemes*, in "Finite Element for Flow Problems, April 4-6, 2005", 2005.

[22] A. DERVIEUX, L. HASCOËT, M. VÁZQUEZ, B. KOOBUS. *Optimization loops for shape and error control*, in "proceedings of the post-SAROD Conference", 2005.

[23] B. KOOBUS, L. HASCOËT, F. ALAUZET, A. LOSEILLE, Y. MESRI, A. DERVIEUX. *Continuous mesh adaptation models for CFD*, in "proceedings of the SAROD Conference", 2005.

[24] C. LAUVERNET, F. BARET, L. HASCOËT, F.-X. LEDIMET. *Improved estimates of vegetation biophysical variables from MERIS TOA images by using spatial and temporal constraints*, in "proceedings of the 9th International symposium on Physical measurements and signatures in remote sensing, ISPMSRS 2005", 2005.

Internal Reports

[25] M. ARAYA-POLO, L. HASCOËT. *Domain of Validity of Derivatives Computed by Automatic Differentiation*, Research Report, n° 5237, INRIA, 2005, http://www.inria.fr/rrrt/rr-5237.html.

[26] A. DERVIEUX, F. COURTY, T. ROY, M. VÁZQUEZ, B. KOOBUS. *Optimization loops for shape and error control*, extended notes after PROMUVAL Short Course on Multidisciplinary Modelling, Simulation and Validation in Aeronautics, Barcelona, Spain, june 28-29, 2004, Research Report, n° 5413, INRIA, 2005, http://www.inria.fr/rrrt/rr-5413.html.

Bibliography in notes

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

[28] I. ATTALI, V. PASCUAL, C. ROUDET. *A language and an integrated environment for program transformations*, research report, n° 3313, INRIA, 1997, http://www.inria.fr/RRRT/RR-3313.html.

[29] A. CARLE, M. FAGAN. *ADIFOR 3.0 overview*, Technical report, n° CAAM-TR-00-02, Rice University, 2000.

[30] 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", Also, Information Processing Society of Japan, Technical Memorandum PL-86-6. Also INRIA research report # 416, 1986, p. 49-89, http://www.inria.fr/rrrt/rr-0416.html.

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

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

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

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

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

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

[37] 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.

[38] A. GRIEWANK. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, SIAM, Frontiers in Applied Mathematics, 2000.

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

[40] P. HOVLAND, B. MOHAMMADI, C. BISCHOF. *Automatic Differentiation of Navier-Stokes computations*, Technical report, nº MCS-P687-0997, Argonne National Laboratory, 1997.

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

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

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

[44] 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.