INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team tropics

Transformations et Outils Informatiques pour le Calcul Scientifique

Sophia Antipolis - Méditerranée

THEME NUM

Activity Report

2007

Table of contents

1. Team

Head of project team
Laurent Hascoët [DR INRIA, HdR]

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

Administrative assistant
Marie-Line Ramfos [TR INRIA]

Staff members
Alain Dervieux [DR INRIA, HdR]

Post-doctoral students
Hicham Tber [(Till August 31st)]

Ph. D. students
Benjamin Dauvergne
Massimiliano Martinelli

Partner scientists
Bruno Koobus [Université de Montpellier 2]

2. Overall Objectives

2.1. Overall Objectives

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

- **AD:** 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, which yields gradients for optimization at a remarkably low cost. The reverse mode of AD 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 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 (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). About AD, our objective is to automatically produce 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 project, and which is becoming 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 : differentiate particular algorithms in a specially adapted manner, compute second-order derivatives, 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

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

Participants: Benjamin Dauvergne, Laurent Hascoët, Massimiliano Martinelli, Valérie Pascual, Hicham Tber.

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

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

checkpointing General trade-off technique, used in the reverse mode of AD, that trades duplicate execution of a part of the program to save some memory space that was used to save intermediate results. Checkpointing a code fragment amounts to running this fragment without any storage of intermediate values, thus saving memory space. Later, when such an intermediate value is required, the fragment is run a second time to obtain the required values.

Automatic or Algorithmic Differentiation (AD) differentiates *programs*. An AD tool takes as input a source computer program P that, given a vector argument $X \in I\!R^n$, computes some vector function $Y = F(X) \in I\!R^m$. The AD tool generates a new source program 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 [36], adjoint problems [31], 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$\times$vector products are so much cheaper than matrix$\times$matrix products. This is the principle of the *reverse mode* of AD.

This turns out to make a very efficient program, at least theoretically [33]. 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.

Unfortunately, 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 [29] 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 [24] and TAPENADE tools use this strategy. Brute-force SA strategy has a linear memory cost with respect to p.

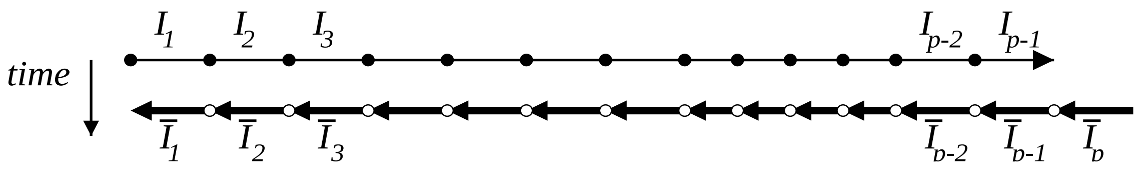


Figure 1. The "Store-All" tactic

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 **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.

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.

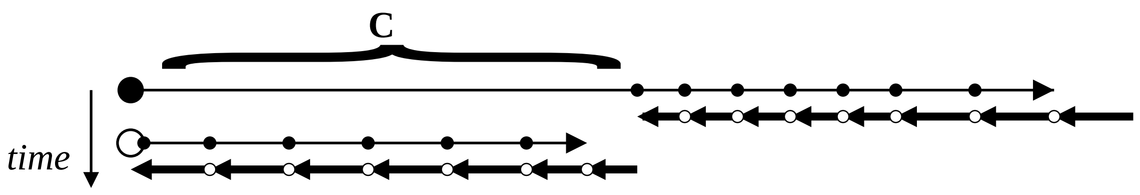


*Figure 2. Checkpointing **C** with the "Store-All" tactic*

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: Benjamin Dauvergne, Laurent Hascoët, Massimiliano Martinelli, Valérie Pascual, Hicham Tber.

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 [22] 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. 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 [25], [34], [23]. 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 [26], 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* [27] 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], [28] 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, Massimiliano Martinelli.

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 [31]), 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 [30] 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 [35] [37] 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 [32], where computation of the adjoint state uses the iterated states in the direct order. Alternatively, most researchers [35] 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 variational 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 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 [38].

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 [contact], Benjamin Dauvergne, Valérie Pascual, Hicham Tber.

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

An 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:

- **Alias analysis:** For any static program transformation, and in particular differentiation, it is essential to have a precise 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 operations.
- **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 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.

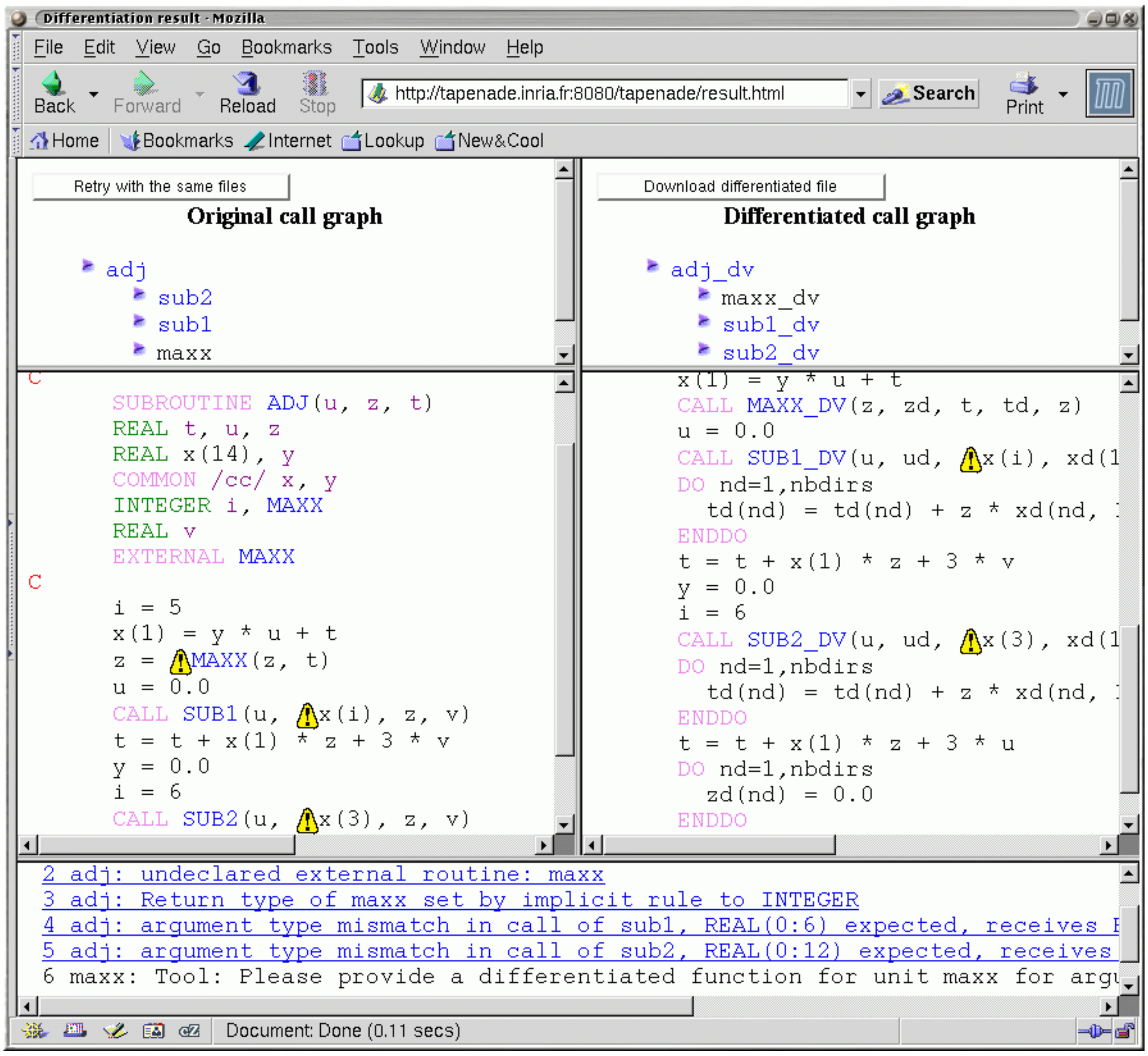
The input languages of TAPENADE are FORTRAN77 and FORTRAN95. Our latest experimental version runs also on C. Thanks to the language-independent internal representation of programs, as shown on figure 4, this still makes a single and only tool, and every further development benefits to differentiation of each input language.

There are 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 lets one specify interactively the routine to differentiate, its independent inputs and dependent outputs. The output interface, shown on figure 3, 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, and WINDOWS-XP platforms.

Figure 4 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.


Differentiation result - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://tapenade.inria.fr:8080/tapenade/result.html
Search Print
Home Bookmarks Internet Lookup New&Cool
Retry with the same files
Download differentiated file
Document: Done (0.11 secs)

Original call graph

```
adj
    sub2
    sub1
    maxx
```

Differentiated call graph

```
adj_dv
    maxx_dv
    sub1_dv
    sub2_dv
```

```
C
      SUBROUTINE ADJ(u, z, t)
      REAL t, u, z
      REAL x(14), y
      COMMON /cc/ x, y
      INTEGER i, MAXX
      REAL v
      EXTERNAL MAXX
C
      i = 5
      x(1) = y * u + t
      z = MAXX(z, t)
      u = 0.0
      CALL SUB1(u, x(i), z, v)
      t = t + x(1) * z + 3 * v
      y = 0.0
      i = 6
      CALL SUB2(u, x(3), z, v)
```

```
      x(1) = y * u + t
      CALL MAXX_DV(z, zd, t, td, z)
      u = 0.0
      CALL SUB1_DV(u, ud, x(i), xd(1
      DO nd=1,nbdirs
        td(nd) = td(nd) + z * xd(nd, 
      ENDDO
      t = t + x(1) * z + 3 * v
      y = 0.0
      i = 6
      CALL SUB2_DV(u, ud, x(3), xd(1
      DO nd=1,nbdirs
        td(nd) = td(nd) + z * xd(nd, 
      ENDDO
      t = t + x(1) * z + 3 * u
      DO nd=1,nbdirs
        zd(nd) = 0.0
      ENDDO
```

```
2 adj: undeclared external routine: maxx
3 adj: Return type of maxx set by implicit rule to INTEGER
4 adj: argument type mismatch in call of sub1, REAL(0:6) expected, receives I
5 adj: argument type mismatch in call of sub2, REAL(0:12) expected, receives
6 maxx: Tool: Please provide a differentiated function for unit maxx for argu
```

Figure 3. TAPENADE *output interface, with source-code-error correspondence*

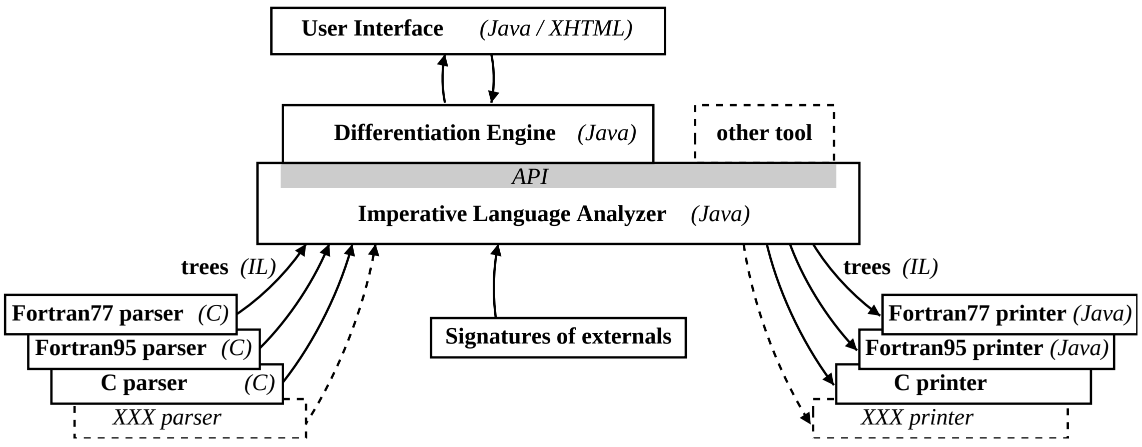


Figure 4. Overall Architecture of TAPENADE

The end-user can 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. Specifically for the reverse mode of AD, TAPENADE lets the user specify finely which procedure calls must be checkpointed or not, to improve the overal performances of the differentiated program.

6. New Results

6.1. Program Analyses to improve Reverse AD

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

Participants: Benjamin Dauvergne, Laurent Hascoët, Valérie Pascual, Hicham Tber.

As described in 3.1, the reverse mode of AD is a very appealing approach to compute gradients, but it suffers from the need to restore intermediate values in the reverse of their original computation order. In our approach, this question is basically solved through storage, whose cost can be very high. We use checkpointing to mitigate this cost, trading extra recomputations for memory space. An efficient storage and checkpointing strategy is really the key to a wide usage of reverse AD in industry. This is why we continue our modelization and experimentation efforts to find the best possible strategies.

Checkpointing can be applied at many places in a program, and we restrict these places to the sites of subroutine calls. Still, for a call tree with n nodes, this accounts for 2^n possible choices for the placement of checkpoints. Following on last year's fine description of the sets of variables that must be saved, Benjamin Dauvergne proposed an approximation of the cost of a given placement of checkpoints, in terms of cpu and maximum memory occupation. He studied the combinatorial problem of finding the placement of checkpoints which is cheapest in cpu, while using less memory than a fixed bound. He found this problem to be NP hard, and proposed a proof of that. He designed a heuristic which returns good results on our example codes. This work was partly done during a one-month visit of Benjamin Dauvergne to Uwe Naumann's team at RWTH Aachen. This work will be described in Benjamin Dauvergne's PhD dissertation.

Laurent Hascoët has continued a scientific collaboration with Jean Utke (Argonne National Laboratory, USA) and Uwe Naumann about using inversion of selected assignments to retrieve temporary values without storing them [15]. The three had a two-weeks meeting at Argonne in september to explore related research directions, such as using graph theory to find optimal placement for storage of individual intermediate variables in an arbitrary control-flow graph.

6.2. Target language extensions in TAPENADE

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

Participants: Laurent Hascoët, Valérie Pascual.

Apart from regularly adapting TAPENADE to Fortran90, Valérie Pascual has devoted most of her time on the adaption to C. The front- and back-ends for C have been developped in the past years, but the analysis and differentiation kernels still remained to be adapted. Although the architecture of TAPENADE was designed as independent from the application language, some problems arised that needed to be solved.

The major development concerned declarations. The problem was already present for Fortran, but was postponed constantly. The internal representation of programs in TAPENADE only keeps statements. Declarations are digested into the Symbol Tables, and then lost. On the way back, the declaration part of the differentiated code, although correct, has lost all ressemblance with the original declarations. As a consequence, the order of declarations, the comments, the placement of include directives, are all lost. This became unbearable with C, with systematic use of include files such as `stdio.h`. Now declarations are kept in the internal representation. Regenerated declarations use the include files and keep the comments from the original program. Generated codes are more readable and often smaller.

Our test base of C programs for differentiation is growing rapidly (50 non-regression tests) and we will soon release the first TAPENADE for C.

6.3. Differentiation of large real applications

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

Participants: Jacques Blum [Université de Nice], Samuel Buis [INRA, Avignon], Benjamin Dauvergne, Laurent Hascoët, Valérie Pascual, Hicham Tber, Arthur Vidard [MOISE team, LMC/IMAG, Grenoble].

We study application of Automatic Differentiation to several very large scientific computation codes. Because of the technical subtleties of AD, differentiation of large codes often requires close collaboration between the end-users and TROPICS. This year, we had such collaboration about

- assimilation of the parameters of vegetation using satellite images (with INRA Avignon),
- estimation of soil parameters (with ANDRA and Jacques Blum's team in Nice)
- inverse problems in Oceanography (with the LOCEAN lab. at Paris 6 and INRIA team MOISE in Grenoble)

The main effort has been put on the Oceanography code OPA 9.0. OPA is developed mainly by the LOCEAN lab. at Paris 6 university. It consist of 80,000 lines of Fortran90. Hicham Tber used TAPENADE to build the tangent and adjoint codes of OPA for a rather complete configuration of OPA called ORCA-2. The tangent and adjoint codes have ben successfully validated. So far these results are described in a tech report [21]. Hicham Tber presented his results at the Grenoble meeting of the LEFE project, which is in charge of the NEMO ocean simulator based on OPA, and to the developers of OPA (LOCEAN lab.) in Jussieu in may.

The two main technical difficulties were the typical length of the simulations (more than 5000 time steps) and the cumulative inaccuracies coming from the adjoint of the Preconditioned Conjugate Gradient (PCG) algorithm which is run at each time step. These were solved using the recent capacities of TAPENADE, i.e. the binomial checkpointing scheme that scales up to several thousands of time steps, and the black-box mechanism to exploit the auto-adjointness of the PCG solver. Figure 5 shows one sensitivity map produced by the TAPENADE-generated adjoint, over 5475 time steps of simulation.

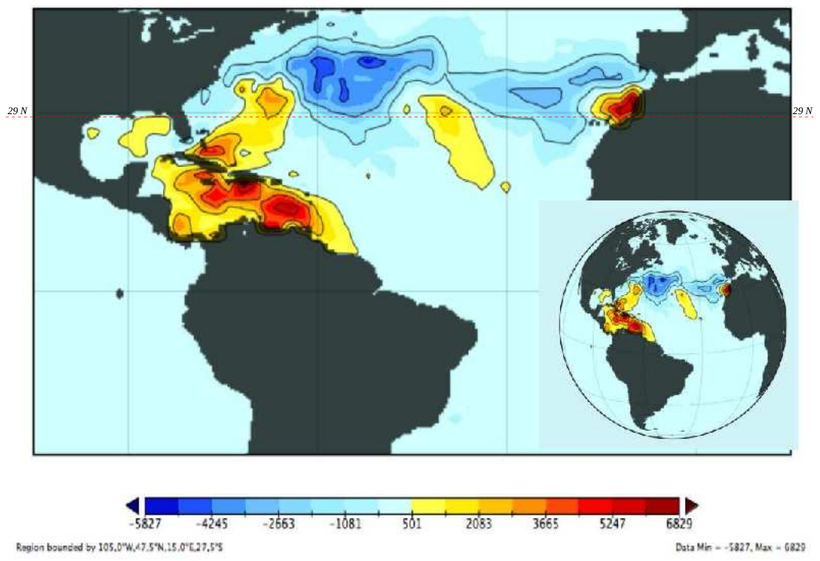

29 N
29 N
-5827
-4245
-2663
-1081
501
2083
3665
5247
6829
Region bounded by 105,0°W,47.5°N,15.0°E,27,5°S
Data Min = -5827, Max = 6829

Figure 5. Sensitivity map of the North Atlantic heat transport at 29^{o}N (dotted line), with respect to changes in the initial surface temperature after one year

6.4. Second Derivatives

Keywords: *Automatic Differentiation*, *Hessian*, *Tapenade*, *gradient*.

Participants: Alain Dervieux, Laurent Hascoët, Massimiliano Martinelli.

Massimiliano Martinelli has been studying 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. ToR wins over ToT only when the number n of input parameters is large enough. We propose an approximation for the threshold value of n. Further research is required to fully compare RoT with ToR.

The ToR approach raises technical questions related to the choice of TAPENADE to store intermediate values on a dynamic stach. We are investigating extensions to TAPENADE, to efficiently handle tangent differentiation of the stack primitives present in the reverse differentiated codes.

This work will be described in Massimiliano Martinelli's PhD dissertation.

6.5. Optimal control

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

Participants: Frédéric Alauzet [GAMMA team, INRIA-Rocquencourt], Francois Beux [Scuola Normale Superiore di Pisa, Italy], Alain Dervieux, Régis Duvigneau [OPALE team], Laurent Hascoët, Bruno Koobus, Massimiliano Martinelli, Youssef Mesri [SMASH team], Mariano Vázquez [Universitat de Girona, Spain].

In industry research groups, simulation is well mastered. The next frontier is optimization. This problem is hard, because the typical number of optimization parameters is high, particularly in CFD optimal shape design. In an industrial context, an accurate discretization of the shape of an aircraft takes hundreds of parameters, hence hundreds of optimization parameters.

To master the enormous computing power required, we focus on the reverse mode 3.1 of AD. The reverse mode, and the subsequent adjoint state, are in fact the most efficient way to get the gradients needed by optimization [13], [12].

In the European project HISAC on supersonic aircrafts (6.7), the state equation cannot be solved accurately without a strong anisotropic mesh adaptation. Therefore, we have to design a new algorithm for the simultaneous solution of shape optimisation and mesh adaptation [19], [20].

In the European project NODESIM, we examine the efficiency problem for solving a large scale adjoint system. Reverse differentiation of the discrete state residual gives us the residual of the discrete adjoint system, but not the explicit matrix that appears in the adjoint equation. Therefore, we resort to a matrix-free solver (GMRES). Faster convergence is obtained by using the first-order (in space) Jacobian of the state equation combined with incomplete factorization ILU(k) [11].

6.6. Management of incertainties

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

Participants: Alain Dervieux, Laurent Hascoët, Massimiliano Martinelli, Régis Duvigneau [OPALE team].

Uncertainties are errors that the engineer cannot reduce by further efforts. One way to take them into account in the process is to model their source by random variables and to apply Monte-Carlo methods to approximate statistical properties of systems output. Since systems can be described by computer-intensive high-fidelity Navier-Stokes models, this strategy can have an unacceptable computational cost. In the European project NODESIM, reduced-order models are obtained by using the first and second derivatives of the high-fidelity models. This is an important application of second-order derivation with TAPENADE [11], [18]. Another way to address uncertainties consists in applying *Robust Optimisation* strategies by adding to a functional some sensitivity terms, again obtained by Automatic Differentiation.

6.7. Control of approximation errors

Keywords: *adjoint*, *mesh adaptation*, *optimization*.

Participants: Frederic Alauzet, Alain Dervieux, Bruno Koobus, Adrien Loseille [GAMMA team, INRIA-Rocquencourt], Massimiliano Martinelli, Youssef Mesri.

This is a joint research between INRIA teams GAMMA (Rocquencourt), TROPICS, and SMASH. Roughly speaking, GAMMA brings mesh and approximation expertise, TROPICS contributes to adjoint methods, and SMASH works on approximation and CFD applications.

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 setting the mesh adaptation problem under purely functional form: 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 metric. Then 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 [16], [13]. In the case of adaptation for Partial Differential Equations, we obtain an Optimal Control problem. Together with project-team GAMMA (Frédéric Alauzet, Adrien Loseille), TROPICS contributes this research to the HISAC IP European project, which involves 30 other partners in aeronautics.

7. Dissemination

7.1. Links with Industry, Contracts

- TROPICS participates in the European IP project HISAC, driven by Dassault Aviation. TROPICS, GAMMA, and SMASH design simulation tools to reduce the sonic boom.
- TROPICS participates in the project EVA-Flo: “Evaluation et Validation Automatique pour le calcul FLOttant”, which is an ANR project accepted this year, 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 this year. Our contribution is to provide the automatic production of the adjoint of OPA (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.
- One license of TAPENADE was sold to the British branch of a Swiss bank. The application is to compute the “greeks”, i.e. the derivatives that occur in option pricing. This application is made under supervision of Mike Giles from Oxford University.
- 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 co-organized with the OPALE team the 42nd conference on Applied Aerodynamics, at INRIA Sophia-Antipolis, on march 19-21.
- Hicham Tber presented his first validated adjoint of OPA during the meeting of the LEFE project in Grenoble, on march 22nd.
- Massimiliano Martinelli and Laurent Hascoët attended the EUCCO’07 conference in Montpellier, on april 2-4. Both made a presentation.
- Laurent Hascoët attended the april 11-12 kick-off meeting of EVA-Flo, research project funded by ANR on the study of truncation errors.
- Laurent Hascoët gave an invited lecture on AD at the ERCOFTAC course on design optimization in Trieste, Italy, on april 18-19.
- Laurent Hascoët is one of the organizers of the euro-AD workshops. This year, one edition took place in Hatfield, UK on may 21-22 and one at INRIA Sophia-Antipolis on november 15-16.
- Benjamin Dauvergne presented his work during the ICIAM’07 conference in Zurich, Switzerland, on july 18-19.
- The TROPICS team hosts the 6th European Workshop on AD at INRIA Sophia-Antipolis on november 15-16. 40 participants. Massimiliano Martinelli and Hicham Tber gave talks.
- Alain Dervieux made several presentation at the West-East High Speed Flow Field Conference 2007 in Moscow, Russia, on november 19-23.
- Laurent Hascoët was on the HDR jury for Renaud Marlet, of INRIA Bordeaux. The work of Renaud Marlet concerns specialization of programs through Partial Evaluation, which is yet another very interesting program analysis and transformation.

- Alain Dervieux organized the annual meeting of the NODESIM EU project at INRIA Sophia-Antipolis on november 26-28. 20 participants. Laurent Hascoët and Massimiliano Martinelli organized a training on AD on the 3rd day.
- Massimiliano Martinelli defended his PhD at the Scuola Normale Superiore di Pisa, Italy, on december 7th. Both Alain Dervieux and Laurent Hascoët served on the Jury.

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. DERVIEUX, L. HASCOËT, M. VÁZQUEZ, B. KOOBUS. *Optimization loops for shape and error control*, in "Recent Trends in Aerospace Design and Optimization", Tata-McGraw Hill, New Delhi, 2005, p. 363-373.

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

[6] 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.

[7] 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.

[8] 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.

[9] L. HASCOËT, V. PASCUAL. *TAPENADE 2.1 user's guide*, Technical report, n^o 300, INRIA, 2004, http://hal.inria.fr/inria-00069880.

[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

Doctoral dissertations and Habilitation theses

[11] M. MARTINELLI. *Sensitivity evaluation in aerodynamic optimal design*, Ph. D. Thesis, Scuola Normale Superiore di Pisa, 2007.

Articles in refereed journals and book chapters

[12] A. DERVIEUX, Y. MESRI, F. COURTY, L. HASCOËT, B. KOOBUS, M. VÁZQUEZ. *Calculs de sensibilité par differentiation pour l'Aérodynamique*, in "ESAIM: PROCEEDINGS", vol. 10, 2007.

[13] A. DERVIEUX, M. VÁZQUEZ, L. HASCOËT, B. KOOBUS. *Optimization loops for shape and error control*, in "CFD Journal", vol. 12, n^{o} 3, 2007.

[14] L. HASCOËT, B. DAUVERGNE. *Adjoints of large simulation codes through Automatic Differentiation*, in "REMN Revue Européenne de Mécanique Numérique / European Journal of Computational Mechanics", To appear, vol. 17, n^{o} 63-86, 2008.

[15] L. HASCOËT, J. UTKE, U. NAUMANN. *Cheaper Adjoints by Reversing Address Computations*, in "Scientific Programming", to appear, 2007.

[16] B. KOOBUS, L. HASCOËT, F. ALAUZET, A. LOSEILLE, Y. MESRI, A. DERVIEUX. *Continuous mesh adaptation models for CFD*, in "CFD Journal", vol. 12, n^{o} 3, 2007.

[17] C. LAUVERNET, F. BARET, L. HASCOËT, S. BUIS, F.-X. LEDIMET. *Multitemporal-patch ensemble inversion of coupled surface-atmosphere radiative transfer models for land surface characterization*, in "Remote Sensing of Environment", to appear, 2007.

Publications in Conferences and Workshops

[18] M. MARTINELLI, A. DERVIEUX, L. HASCOËT. *Strategies for computing second-order derivatives in CFD design problems*, in "Proc. of West-East High Speed Flow Field Conference, Moscou, Russia", nov. 19-22 2007.

[19] Y. MESRI, F. ALAUZET, A. DERVIEUX. *A strongly coupled mesh adaptative optimal shape algorithm*, in "Proc. of NMFD-ICFD, Reading, UK", march 2007.

[20] Y. MESRI, F. ALAUZET, A. DERVIEUX. *Coupling optimum shape design and mesh adaptation for sonic boom reduction*, in "Proc. of 42th CAA-AAAF, Sophia-Antipolis (F)", march 2007.

Internal Reports

[21] M.-H. TBER, L. HASCOËT, A. VIDARD, B. DAUVERGNE. *Building the Tangent and Adjoint codes of the Ocean General Circulation Model OPA with the Automatic Differentiation tool TAPENADE*, Research Report, n^{o} 6372, INRIA, 2007, http://hal.inria.fr/inria-00192415.

References in notes

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

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

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

[25] 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://hal.inria.fr/inria-00076140.

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

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

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

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

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

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

[32] 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.

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

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

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

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

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

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

[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.