Generating efficient derivative code with TAF: Adjoint and tangent linear Euler flow around an airfoil

R. Giering [a], T. Kaminski [a,*], T. Slawig [b]

[a] *FastOpt, Martinistr. 21, 20251 Hamburg, Germany*

[b] *Technische Universität Berlin, Straße des 17. Juni 136, D-10623 Berlin, Germany*

Abstract

FastOpt's new automatic differentiation tool TAF is applied to the two-dimensional Navier-Stokes solver NSC2KE. For a configuration that simulates the Euler flow around a NACA airfoil, TAF has generated the tangent linear and adjoint models as well as the second derivative (Hessian) code. Owing to TAF's capability of generating efficient adjoints of iterative solvers, the derivative code has a high performance: Running both the solver and its adjoint requires 3.4 times as long as running the solver only. Further examples of highly efficient tangent linear, adjoint, and Hessian codes for large and complex three-dimensional Fortran 77-90 climate models are listed. These examples suggest that the performance of the NSC2KE adjoint may well be generalised to more complex three-dimensional CFD codes. We also sketch how TAF can improve the adjoint's performance by exploiting self-adjointness, which is a common feature of CFD codes.

Key words:
PACS: 89.80, 02.70, 07.05.T

1 Introduction

Many applications in Computational Fluid Dynamics (CFD) do benefit from availability of sensitivity information. Examples span the range from multi-

* Corresponding author.
Email addresses: Ralf.Giering@FastOpt.com (R. Giering), Thomas.Kaminski@FastOpt.com (T. Kaminski), slawig@math.TU-Berlin.de (T. Slawig).

disciplinary shape design of airfoils or turbomachinery blades to modelling of atmosphere or ocean dynamics [1–3]. Sensitivities quantify the impact of a change in certain control variables on particular target quantities of interest. In aerodynamics or aeroacoustics applications, lift and drag or kinetic energy are examples of such target quantities, and the control variables define the shape of the object under consideration. In atmosphere and ocean modelling, typical target quantities are integrals of the large-scale circulation or the difference between observations and their model-simulated counterparts. Typical control variables are the initial state, boundary values, or parameters in the model formulation.

In some applications the sensitivity information is interpreted directly. In others this interpretation is done by an optimisation algorithm, which iteratively exploits sensitivity information to vary the control variables in order to improve the value of the target quantity. Second-order sensitivity (Hessian) information [4] is important to speed up this search process and to analyse robustness of the solution [5,6].

There are different strategies of deriving sensitivity information. A first approach, also called continuous approach, applies perturbation theory [7] to the model: the model equations are linearised, discretised and the tangent linear model is coded. For most problems, however, the desired number of control variables is much larger than that of the target quantities. For many of these problems, sensitivity computation is only computationally feasible via an adjoint formulation of the model equations [8]. The adjoint equations are then discretised and coded, which yields the adjoint model.

An alternative strategy of obtaining sensitivity information applies Automatic Differentiation (AD) (see [9] and references therein) directly to the code of the model: To generate the derivative code (tangent linear or adjoint model), the model code is decomposed into elementary functions, which more or less correspond to the individual statements in the code. These elementary functions are differentiated (this derivative is also called local Jacobian). The derivative code multiplies these local Jacobians, which, according to the chain rule, yields the derivative of the composite function. As opposed to derivative approximation by divided differences (also known as numerical differentiation), AD provides sensitivity information that is accurate within round-off error.

Like the continuous approach, AD can construct both tangent linear and adjoint models. The tangent linear model uses the order, in which the model evaluates the statements, to evaluate the product of their Jacobians. The adjoint model does this evaluation in reverse order. In AD terminology, the tangent linear model operates in forward mode and the adjoint model operates in reverse mode. Similar to the finite difference approximation, the computational resources needed in forward mode increase with the number of control

variables. In reverse mode, they are roughly proportional to the number of target quantities, but virtually independent of the number of control variables. The availability of the reverse mode is another major advantage of AD over the finite difference approximation.

Applying the continuous approach requires the choice of discretisation schemes for both the model equations and the adjoint equations. Typically, in the discretisation step, the adjoint relation is only valid in an approximate sense. Consequently, the sensitivity information that is provided by the adjoint code is not fully consistent with the actual sensitivity of the model code. This inconsistency, which does not occur when using AD, can be problematic in an optimisation context [10]. Also, for particular applications, the rigorous derivation of the adjoint equations that form the basis of the continuous adjoint approach can become a cumbersome piece of analysis [11,8]. For second derivatives this analysis gets even more complex [4]. By contrast, using AD to construct the adjoint code avoids this analytical effort at all.

AD can be carried out by hand or by an AD tool (e.g. [12–17] see also http://www.autodiff.org). Applying an AD tool restricts the effort of development and maintenance to the model itself: based on the model code, the adjoint and tangent linear models as well as the Hessian code can be generated and maintained automatically. Especially for models under development, this constitutes a significant advantage and calls for an AD tool as integral component of a state-of-the-art modelling system [18,19]. By contrast, the continuous approach requires coding and maintaining the derivative code by hand. For CFD codes written in Fortran 77, AD-tools have been applied to generate many tangent linear codes (e.g. [20–25]) and few adjoint (e.g. [26,28]) and Hessian (e.g. [29]) codes .

This paper introduces the relatively new AD tool Transformation of Algorithms in Fortran (TAF, [16]), which handles Fortran 77-95 code. For almost a decade, TAF's predecessor TAMC ([17]) has been generating tangent linear and adjoint models, as well as Hessian code of an ever increasing number of, among others, large models of atmosphere and ocean dynamics. The performance of the derivative code generated by TAMC and TAF is very high, which is crucial for the feasibility of most applications.

Recently, the first TAMC and TAF applications to Navier-Stokes solvers for shape design and instantaneous control have been completed ([30–34]). Using an example of a two-dimensional Navier-Stokes solver (NSC2KE, [35]), this paper describes how TAF can be applied to generate highly efficient adjoint and tangent linear and Hessian code. As many other CFD codes, NSC2KE uses an iterative solver. We show how to trigger generation of efficient adjoint code for iterative solvers.

The remainder of this paper is organised as follows: We first give a brief introduction to TAF, followed by a description of TAF's handling of iterative solvers. Next we describe NSC2KE and its differentiation. We then compare the performance of NSC2KE's derivative code to that of further TAF applications.

2 TAF overview

Transformation of Algorithms in Fortran (TAF, [16]) is an AD tool for Fortran 77-95 programs. TAF operates as a source-to-source transformation tool. That is, from a given Fortran program, which evaluates a function, TAF generates a second Fortran program, which evaluates the function's derivative (gradient or Jacobian). TAF generates both forward and reverse mode derivative codes, i.e. tangent linear and adjoint models. In each mode TAF can generate code to evaluate Jacobian times vector products or the full Jacobian. Second order derivative (Hessian) code is generated by invoking TAF twice. Typically, the most efficient strategy of obtaining second derivative information for a scalar valued target quantity is the so-called forward over reverse mode of AD: TAF is invoked to generate the adjoint code, which afterwards is resubmitted to TAF to be differentiated in forward mode.

Another TAF feature is Automatic Sparsity Detection (ASD), i.e. efficient determination of the sparsity structure of the Jacobian. This sparsity information can be important, because the Jacobian's sparsity pattern can be exploited to render the evaluation of the Jacobian more efficient.

Recent TAF enhancements include basic support of parallel programming, namely the Message Passing Interface (MPI) and OpenMP, as well as a mode for generation of a divided adjoint, which allows interruption and restart of the adjoint model run (see [36] for details).

TAF performs an analysis of the data flow in the code to be differentiated, which determines the active/passive variables. Active variables are all variables that depend on the control variables and have influence on the target quantities. All non active variables are called passive variables. Required variables are all variables whose values are needed to evaluate the local Jacobian. For example, in time integrations of non-linear systems the trajectory is part of the required variables. In case there are pieces of source code missing (black box routines), e.g. for library routines, the user can provide the relevant data flow information via TAF flow directives. TAF flow directives are also applied to include available derivative code into TAF generated code, which is extremely useful, e.g., in case of self-adjoint routines. We will show an example in section 3.

The current TAF version (1.4) marks an important progress as compared to the final version of TAF's predecessor TAMC. TAF is much more robust, since we have identified and avoided a large number of TAMC bugs and problems. Owing to a superior internal structure, the derivative code generation was speeded up considerably and uses less memory: Differentiating the 100,000 lines of the MIT GCM [37] on a Linux-PC with Athlon XP 1600 MHz processor takes less than a minute. Furthermore, TAF supports the Fortran 77-95 language standard. As of May 2002, operator overloading, pointers, named arguments and generic functions are still excepted. Also there are a few minor restrictions in handling allocatable arrays (complex allocation/deallocation sequence with varying size for the same array) and derived types (when a component is itself a derived type). By contrast, TAMC is restricted to Fortran 77 with only a few basic Fortran 90 extensions. Also, for the reverse mode, TAF can handle more complex control flows than TAMC, which it normalises and then differentiates [38]. Another important advantage of TAF is that it is maintained, continuously improved, and customised for its users. For instance, the number of exceptions and restrictions is constantly decreasing.

3 Efficient adjoints of iterative solvers

In order for adjoint code to achieve a high performance, it is essential to provide required variables efficiently. In tangent linear code, required variables are easily provided by integrating the model with the derivative code. For adjoint code, it is much more complicated to provide required variables since they are needed in reverse order as compared to the order of their computation in the model. There are two ways of providing a required variable to an adjoint integration:

(1) **storing/reading:** The required values are stored on disk or in memory during an initial model integration and read during the adjoint integration.
(2) **recomputation:** The required values are recomputed by inserting a suitable fraction of the model code into the adjoint code.

For large-scale applications, storing/reading is usually prohibitively disk or memory consuming. By default, TAF applies recomputation, for which it uses an advanced version of the efficient recomputation algorithm ERA described in [39]. For large-scale applications, however, relying only on recomputation is too expensive in terms of CPU time. Most efficient adjoint code always uses a balance of recomputation and storing/reading [40]. In its log file, TAF reports any extensive recomputation. The user can easily trigger storing/reading of selected required values by inserting TAF store directives. All necessary bookkeeping is arranged automatically by TAF. We will show examples of store

directives in section 4.

The CFD code NSC2KE [35] uses a two-dimensional Navier-Stokes solver that integrates the simulated system to a steady flow. The steady flow is insensitive to a change in the initial flow, unless the integration is terminated before convergence. TAF cannot know at compile time whether a loop computes a converging sequence. It will detect a data flow dependence between the initial flow and the steady flow. In the non-linear case, this will render the integration's entire flow trajectory a part of the required variables, such that it has to be provided to the adjoint integration. This slows down the adjoint integration considerably. Christianson [41,42] came up with an alternative adjoint formulation, which is based on the convergence assumption. Instead of the entire flow trajectory, Christianson's formulation only requires the final steady flow. TAF is capable of generating the corresponding alternative adjoint code [40]. The user triggers this by inserting a TAF iteration directive, which declares the respective loop to be converging. The loop must be either a `do` or a `do-while` loop. We will show an example in section 4.

Time integrations that converge to a steady state are merely one of the applications for the efficient alternative adjoint code of converging sequences. Another common example are iterative solvers for algebraic equations.

In case of self-adjoint routines, it is highly efficient and common practise in hand coding to call the original routine in the adjoint code. An operator A acting in a finite dimensional Euclidean space with inner product $[\cdot, \cdot]$ is called self-adjoint if $[x, Ay] = [Ax, y]$ for any elements x and y of that space. A self-adjoint routine is an implementation of a self-adjoint operator.

TAF supports the reuse of self-adjoint routines in the adjoint. Hinze and Slawig [32] took advantage of this feature, when applying TAF to generate the adjoint of their suboptimal control algorithm [43] for the unsteady Navier-Stokes equation. The state equation in every optimisation step of the algorithm is of quasi-Stokes type and thus linear. Moreover it can be equivalently written in a self-adjoint form. Therefore the model solver itself can be used in the adjoint code. Generation of the proper interface in the calling unit of the adjoint code is triggered by only 5 TAF flow directives. These flow directives provide all information needed in TAF's data flow analysis and adjoint code generation phases. The quasi-Stokes solver is a subroutine with the following parameter list:

```
subroutine qst(aup,agp,y,scalgp,scalup,du,yh,irw,dt,1.d0,dt,stime)
```

where `yh` is the input and `y` the output variable. The TAF flow directives have the following form:

```
c$taf subroutine qst input =  1,2,  4,5,6,7,8,9,10,11,12
```

```
c$taf subroutine qst output =     3
c$taf subroutine qst active =     3,      7
c$taf subroutine qst depend = 1,2,  4,5,6,  8,9,10,11,12
c$taf subroutine qst adname = qst
```

As all TAF directives, the flow directives start with the string (sentinel) `c$taf` in Fortran 77 or `!$taf` in Fortran 90-95, respectively, i.e. they are comments for the Fortran compiler. The flow directives name the routine the flow information refers to. The numbers in the first four directives refer to the position of variables in the subroutine's parameter list. The first four directives name input variables, output variables, active variables and required variables, respectively. The final directive provides the name to be used by the adjoint subroutine call, which in the generated code looks as follows:

```
call qst(aup,agp,adyh_h,scalgp,scalup,du,ady,irw,dt,1.d0,dt,stime)
adyh = adyh + adyh_h
```

As compared to the active input and output variables `yh` and `y` the corresponding adjoint variables `adyh_h` and `ady` have swapped their positions in the argument list, where `adyh_h` is an auxiliary array. In general it is not save to generate the call which directly has `adyh` as input argument, because its value might be overwritten by `qst`. The positions of the required variables remain unchanged. Note that this strategy of handling self-adjointness is restricted to routines with only one active input and one active output variable.

The MITGCM uses a CG solver, which is also self-adjoint. A set of flow directives similar to the one above has been employed to reuse the solver in the TAF generated adjoint [44]. The performance gain via flow directives is not restricted to self-adjoint routines. The strategy can be applied whenever the adjoint of a routine is available in the model code (or can be easily extracted from it). Another standard example is the Fourier Transform (or Fast Fourier Transform, FFT), which (with proper normalisation) is a unitary operator. This means its adjoint equals its inverse. These properties have been exploited for the adjoint of the NASA-DAO finite volume GCM's [45–47] FFT. With flow directives similar to those shown above, TAF has generated an adjoint which reuses the model's FFT subroutine.

4 Differentiation of the Navier-Stokes Solver NSC2KE

NSC2KE is a mixed finite volume-finite element Galerkin Computational Fluid Dynamics (CFD) model implemented in Fortran 77 ([35]). It simulates a two-dimensional flow on an unstructured grid. The model provides the options to solve the Euler or the full compressible Navier-Stokes equation with a k-ε

turbulence model. In the Euler part Roe-, Osher-, or kinematic solvers can be chosen. The time integration is based on a fourth-order Runge-Kutta scheme.

NSC2KE has already served as a test code for other AD tools. Mohammadi et al. [26] and Ulbrich [27] have generated adjoint versions of NSC2KE using Odyssée [15] and TAMC [40], respectively. Hovland et al. [21] and Slawig [22] have generated tangent linear models with ADIFOR [13]. We have generated the tangent linear, adjoint, and Hessian codes for a configuration using the Euler model with Roe Solver, which simulates the steady-state flow around the wing NACA 0012. The grid has 801 vertices, 1516 triangles, and 2317 edges. In this configuration, the model converges in less than 500 time steps. We use the free stream Mach number and the angle of attack as control variables, and the lift as target quantity.

We have slightly rearranged the model structure by splitting initialisation and postprocessing off the main loop. The main loop is executed in a subroutine, which has the control variables and the target quantity as arguments. The structure of the main loop has been transformed to a `do` loop, which schematically looks as follows:

```
c$taf init tape1 = static, 1
c$taf loop = iteration, ua, un, pres
      do ktout = 1, ktmax
         kt = kt0 + ktout
c$taf store pres,reyturb,ua,t = tape1
            call caldtl(dtmin,dt)
c$taf store reylam,dtl        = tape1
            call runge( kt0, som )
      end do
```

As described in section 3 the TAF iteration directive right before the `do` loop triggers generation of the efficient alternative adjoint loop. The variable names `ua, un`, and `pres` comprise the flow, i.e. their values converge. The TAF init directive triggers generation of a tape termed `tape1`, which is static (i.e. realised in a common block) and only needs to hold 1 record of each of the variables `pres,reyturb,ua`, and `t`. Note that TAF also offers tapes in dynamic memory or on file. `tape1` records the values that the variables named in the two TAF store directives converge to. They are required variables to the adjoint Runge-Kutta solver in subroutine `runge`. To trigger the adjoint model's entire storing/reading scheme there are two more tapes (two more TAF init directives) and 14 TAF store directives not shown here. The model uses a Newton solver for the boundary conditions, which also computes a converging sequence. After changing the loop to a `do-while` structure, another TAF iteration directive triggers generation of an efficient adjoint for this Newton solver. All tapes for storing required variables have been realised as common

blocks in core memory. This demonstration configuration uses about 1.5 MB of memory. Including the tapes for the required variables, the adjoint uses about 3.2 MB of memory and the Hessian code about 5.0 MB.

Slight rearrangements of the initial model code at a few places further helped to support the TAF analysis: An `if-then` structure was extended by an `else` branch, and the code for reading of restart fields was removed. In a nested `if` condition construct, logical operators have been rearranged. At another place, three `if-then` statements with mutually excluding conditions have been transformed to a single `if-then-else` structure. These modifications helped to save recomputations of required values and thus provided an additional gain of efficiency. We would like to point out that these preparations affect only a few rather obvious parts of the code, which were mostly indicated by the TAF log file (see section 3).

At an additional location, we speeded up a slow model code fragment, which does and undoes a scaling of the model state, by introducing a loop over the state variables. This modification also speeded up the corresponding derivative code.

Without comments, the model comprises 2,485 lines of Fortran code the tangent linear model comprises 4,232 lines, the adjoint model 7,472 lines, and the Hessian code 19,384 lines. The gradient computations by the tangent linear and the adjoint model have been verified against each other and against finite differences of model evaluations. In double precision, with a finite difference interval of 10^{-8}, the finite difference approximations of the angle of attack and the Mach number are accurate to less than a permil. The Hessian code has been verified against finite differences of the adjoint. Again, finite differences are accurate to less than a permil.

As an example, Figure 1 depicts the sensitivity of the lift with respect to the vertical velocity of the flow. This illustrative intermediate result of the back-propagation of sensitivities from the lift to the control variables is a byproduct of the adjoint integration.

5 Performance

This section presents the performance of the derivative code of NSC2KE, and, for comparison, that of further TAF applications. Table 1 summarises the characteristics of the respective models. The first two columns name the model and give a reference for the model. Columns 3 and 4 indicate the number of the model's source code lines without comments and the programming language. Column 5 characterises the main loop, where "steady" refers to convergence to

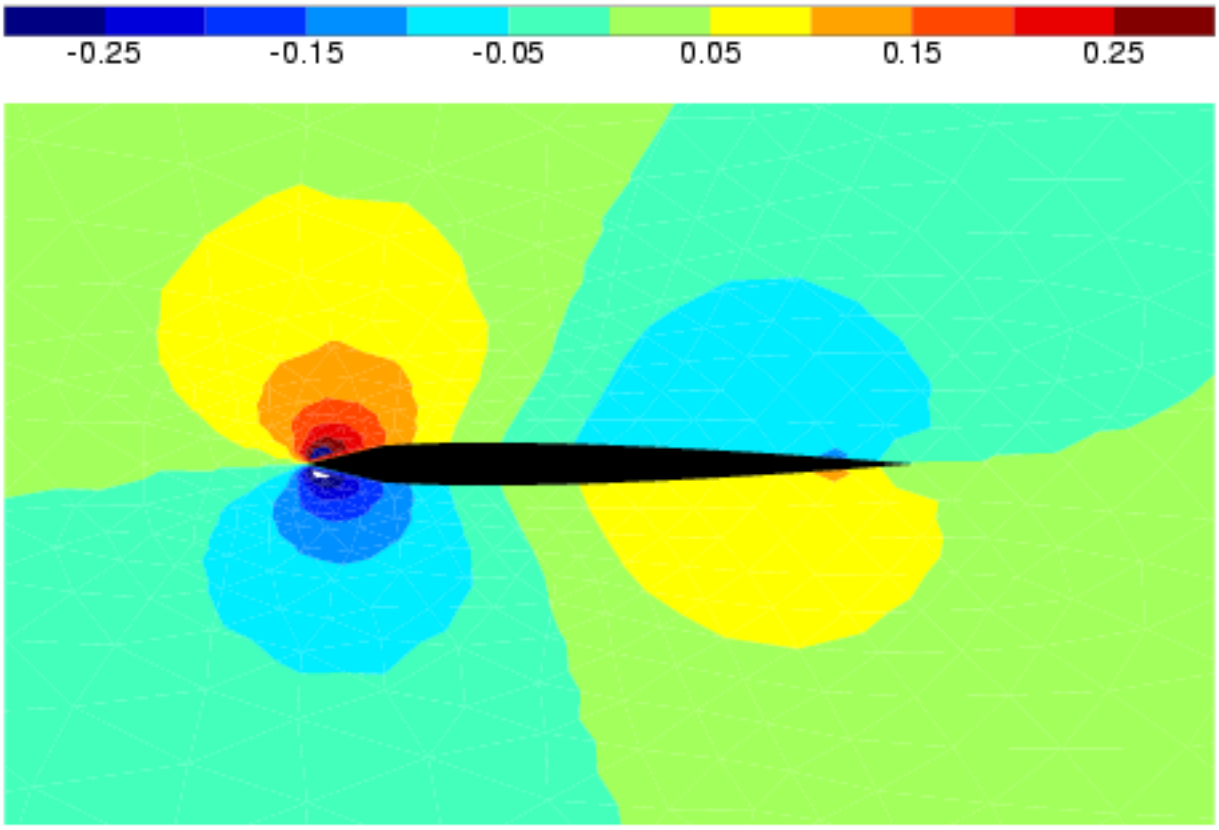

Fig. 1. Example: Adjoint sensitivity of lift to vertical velocity of the flow. Negative sensitivities are blue, positive sensitivities range from yellow to red

a steady (or stationary) flow and "evolving" refers to the forward integration of a time-evolving system. Columns 6 to 8 indicate whether the tangent linear model (TLM), the adjoint (ADM) or Hessian code have been generated. The last column lists the references for the derivative codes.

NSC2KE has been presented in detail in the section 4. The Navier-Stokes solver by Hinze [43] solves a time-dependent optimisation problem for the Navier-Stokes equation by a so-called instantaneous strategy: In every time step, a stationary control problem is solved approximately. The generated temporal sequence of controls turns out to be an effective suboptimal control. The code is mostly Fortran 77 with about 450 lines for the stationary solver (excluding the self-adjoint routine described in section 3). Carbon-BETHY is a coupled model consisting of a reduced version of the terrestrial biosphere model BETHY [48] and the atmospheric transport model TM2 [55] as represented by its Jacobian matrix [56]. The Modular Ocean Model version 3 (MOM3, also known as Bryan-Cox model or GFDL model, see [50]) and the MITGCM [37] are models of the three-dimensional general oceanic circulation. The biomag

Table 1
Overview of TAF applications

Model	Model Ref	Lines	Language	Main Loop	TLM	ADM	HES	AD Ref
NSC2KE	[35]	2,500	F77	steady	yes	yes	yes	
NS-Solver	[43]	450	F77	steady	-	yes	-	[32]
Carbon-BETHY	[48]	5,400	F90	evolving	yes	yes	yes	[49]
MOM3	[50]	50,000	F77	evolving	yes	yes	-	[51]
MITGCM	[37]	100,000	F77	evolving	yes	yes	yes	[52,44]
Biomag code	[53]	83	F77	-	yes	yes	-	[54]
NASA-DAO	[45–47]	87,000	F90	evolving	yes	yes	-	

Table 2
Performance of TAF generated adjoint models

Model	RelCPU	Handling
NSC2KE	3.4	iteration directive
NS-Solver	2.0	flow directives
Carbon-BETHY	3.6	2 level checkpointing
MOM3	4.6	2 level checkpointing
MITGCM	5.5	3 level checkpointing
Biomag code	3.1	
NASA-DAO	7.1	2 level checkpointing

code simulates the magnetic field from a density distribution in a human head (see [53] for details). The NASA-DAO finite volume GCM ([45–47]), is a full three-dimensional model of the atmospheric circulation. Adjoint and tangent linear code have only been generated for the GCM's dynamical core. The Hinze solver, MOM3, and the MITGCM have been differentiated with TAMC first, and the respective groups have since switched to TAF. In a pre-processing step, most of the orginal MOM3 code's Fortran 90 statements had been replaced by corresponding Fortran 77 statements, in order to render the code TAMC-compliant.

Table 2 shows the performance of the ADMs of the models listed in 1. All performance ratios refer to the gradient of a scalar valued target quantity. The first column names the model, and the second column gives the ratio of the CPU times for an evaluation of gradient plus function to an evaluation of the function only.

The third column indicates the way the main loop is handled in order to achieve memory/disk-efficient code. A TAF iteration directive has triggered generation of efficient alternative adjoint code for NSC2KE, and the adjoint of the Hinze solver exploits self-adjointness via TAF flow directives (see section 3). For time-evolving systems TAF can automatically generate a so-called checkpointing scheme [57], in its linear form [58]: During an initial model integration, the state of the system is periodically saved on a first tape (outer tape) in intervals of a fixed number of time steps. The adjoint integration then goes backwards interval by interval. For each of these intervals, first the model is integrated over the interval, and required variables are stored on a second tape (inner tape). During the corresponding adjoint integration over this interval, the required values are then read from the inner tape. This inner tape is reused for the next interval, which considerably reduces disk/memory requirements. Carbon-BETHY, for instance, typically simulates a period of 20 years with one checkpoint per month. The inner tape has about 50 MB, which fits well in the memory of a Linux PC, while the outer tape of about 140 MB fits well on the machine's hard disk. Without checkpointing one would need 50*21*12 MB, i.e. about 12 GB of tape space. The cost of this scheme, however, is (in total) about one additional model integration. Automatic generation of a checkpointing scheme is triggered, after splitting up the main time stepping loop into an inner and an outer loop, by inserting a TAF store directive plus TAF initialisation directives for both the inner and outer tape. A more elaborate description is given in [58]. The scheme sketched above uses two-level checkpointing. Some applications call even for a three-level checkpointing scheme, in which the main loop is split into three nested loops and three tapes are used. Triggering its generation works analogously to the two-level case. For example, the performance ratio of 5.5 for the MITGCM with three-level checkpointing thus means that a short integration (without the need of a checkpointing scheme) has a performance ratio of about 3.4 for the run times of adjoint model plus model to model only.

For the Navier-Stokes solver by Hinze, an ADM has also been hand-coded via the continuous approach. The relative performance of that ADM is about 1.8 ([32]). Note that the ADM of the NASA-DAO finite volume GCM is not yet fully optimised for performance.

Table 3 shows the performance of the tangent linear models of the various models named in the first column. The second column gives the number of control variables. Except for the biomag code, the performance has been measured in a Jacobian times vector mode, which is equivalent to having one control variable. The data locality of TAF-generated tangent linear code increases with the number of control variables. Thus, on cache-based machines, the number of cache misses per control variable is decreasing. Consequently, also the CPU time per control variable decreases with the number of control variables.

Table 3
Performance of TAF generated tangent linear models

Model	# of control variables	RelCPU
NSC2KE	1	2.4
Carbon-BETHY	1	1.5
MITGCM	1	1.8
Biomag code	1,098	548
NASA-DAO	1	2.7

The Hessians of NSC2KE, Carbon-BETHY, and the MITGCM are computed in forward over reverse mode (see section 2). Evaluating the product of the NSC2KE's Hessian with a vector takes 9.8 times the CPU time of a model run. For the MITGCM, that ratio is 11.0. The full Hessian of Carbon-BETHY has been computed in chunks of five columns per run to fit in the available memory. Per column, the computation takes 2.5 times the CPU time of a model run. When computing only two columns per run, this per column CPU time ratio goes up from 2.5 to 5.1. This has two reasons: First, in this forward over reverse mode, the adjoint integration constitutes a fraction of the Hessian computation which is independent of the number of columns. Second, as described above for the tangent linear code, when increasing the number of columns per run, the increased data locality further increases the "per-column performance" of the Hessian code.

6 Conclusions and Perspectives

We have applied TAF to the Navier-Stokes solver NSC2KE. For a configuration that simulates the Euler flow around a NACA airfoil, TAF has generated the solver's tangent linear and adjoint models. The high performance ratios of 2.4 for the tangent linear model, 3.4 for the adjoint model, and 9.8 for the Hessian code are in the range of TAF applications, e.g. to large-scale models of ocean and atmosphere dynamics. It has been demonstrated that TAF generates efficient alternative adjoints of iterative solvers. Furthermore, we have sketched how TAF can exploit self-adjointness to improve the performance of the adjoint code. While NSC2KE is written in Fortran 77, a number of recent applications demonstrate TAF's additional capability of handling Fortran 90 codes.

Future TAF developments will be directed towards further improving the performance of the generated derivative code, robustness of the tool, and its user friendliness. In practise, this means we will be working on topics such as

further improving our transformation algorithms, following future extensions of the language standard, enhancing TAF interactions with common parallel programming extensions/libraries, as well as improving our algorithms to reduce complex control flows, and introducing an automatic decision between recomputing and storing/reading.

Another future FastOpt project is to transfer the TAF concepts from Fortran to C and build an AD tool for programs written in C, which will be called Transformation of Algorithms in C (TAC).

Acknowledgments

FastOpt wishes to thank the following colleagues: Wolfgang Knorr, Peter Rayner, and Marko Scholze for the pleasant collaboration in composing a Carbon Cycle DataAssimilation and prediction System (CCDAS), which includes the derivative code of Carbon-BETHY. Ricardo Todling, S-J Lin and their colleagues at the Data Assimilation Office for the good collaboration in our joint work on the derivative code of the NASA-DAO GCM. Eli Galanti and Eli Tziperman for providing the performance ratios of their MOM adjoint. Ralf Giering wishes to thank the ECCO consortium, in whose framework the MIT GCM was differentiated, for their collaboration. The authors wish to thank Bijan Mohammadi for providing his CFD code NSC2KE and Johannes Werner for proofreading the manuscript.

References

[1] A. Jameson, Aerodynamic design via control theory, J. of Scientific Computing 3 (1987) 233–260.

[2] O. Talagrand, P. Courtier, Variational assimilation of meteorological observations with the adjoint vorticity equation – Part I. Theory, Q. J. R. Meteorol. Soc. 113 (1987) 1311–1328.

[3] Z. Sirkes, E. Tziperman, C. W. Thacker, Combining data and a global primitive equation ocean general circulation model using the adjoint method, in: P. Malanotte-Rizzoli (Ed.), Modern approaches to data assimilation in ocean modeling, Elsevier, 1996, pp. 119–145.

[4] F.-X. LeDimet, I. Navon, D. Daescu, Second order information in data assimilation, Monthly Weather Review 130 (3) (2002) 629–648.

[5] J. Reuther, Aerodynamic Shape Optimization Using Control Theory, Ph.D. thesis, University of California, Davis, USA (1996).

[6] E. Arian, S. Ta'asan, Analysis of the Hessian for aerodynamic optimization: inviscid flow, Advances in Engineering Software 28 (7) (1999) 853–877.

[7] T. Kato, Perturbation theory for linear operators, Springer, Berlin, 1966.

[8] G. I. Marchuk, Adjoint Equations and Analysis of Complex Systems, Kluwer, Dordrecht, 1995.

[9] A. Griewank, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, SIAM, Philadelphia, 2000.

[10] P. Shah, Application of adjoint equations to estimation of parameters in distributed dynamic systems, in: A. Griewank, G. F. Corliss (Eds.), Automatic Differentiation of Algorithms: Theory, Implementation, and Application, SIAM, Philadelphia, Penn., 1991, pp. 181–190.

[11] M. Reed, B. Simon, Methods of modern mathematical physics. I. Functional analysis, Academic Press, New York, 1980.

[12] C. Bischof, L. Roh, A. Mauer, ADIC — An extensible automatic differentiation tool for ANSI-C, Software: Practice and Experience 27 (12) (1997) 1427–1456.

[13] C. H. Bischof, A. Carle, P. Khademi, A. Mauer, ADIFOR 2.0: Automatic Differentiation of Fortran 77 Programs, IEEE Computational Science & Engineering 3 (3) (1996) 18–32.

[14] A. Griewank, D. Juedes, J. Utke, ADOL-C, a package for the automatic differentiation of algorithms written in C/C++, ACM Trans. Math. Softw. 22 (2) (1996) 131–167.

[15] N. Rostaing, S. Dalmas, A. Galligo, Automatic differentiation in Odyssée, Tellus 45A (1993) 558–568.

[16] FastOpt, Transformation of Algorithms in Fortran, http://www.FastOpt.com. URL `http://www.FastOpt.com`

[17] R. Giering, Tangent linear and Adjoint Model Compiler, http://www.autodiff.com/tamc.
URL `http://www.autodiff.com/tamc`

[18] A. Adcroft, J.-M. Campin, P. Heimbach, C. Hill, J. Marshall, The MITgcm, Online documentation, Massachusetts Institute of Technology, USA (2002).
URL `http://mitgcm.org/sealion/online_documents`

[19] S. M. Griffies, M. J. Harrison, R. C. Pacanowski, A. Rosati, The FMS MOM4-beta User Guide, Tech. rep., NOAA/Geophysical Fluid Dynamics Laboratory (2002).
URL
`http://www.gfdl.noaa.gov/~lat/fms_public_release/public_manual_fms/mom4beta_manual.htm`

[20] A. Carle, L. Green, C. H. Bischof, P. Newman, Applications of automatic differentiation in CFD, in: Proceedings of the 25th AIAA Fluid Dynamics Conference, AIAA Paper 94-2197, American Institute of Aeronautics and Astronautics, 1994.

[21] P. D. Hovland, B. Mohammadi, C. H. Bischof, Automatic differentiation of Navier-Stokes computations, Tech. Rep. MCS-P687-0997, Argonne National Laboratory (1997).

[22] T. Slawig, Domain optimization of a multi-element airfoil using automatic differentiation, Advances in Engineering Software 32 (2001) 225–237.

[23] S. A. Forth, T. P. Evans, Aerofoil Optimisation via AD of a Multigrid Cell-Vertex Euler Flow Solver, in: G. Corliss, C. Faure, A. Griewank, L. Hascoët, U. Naumann (Eds.), Automatic Differentiation: From Simulation to Optimization, Computer and Information Science, Springer, New York, 2001, Ch. 17, pp. 153–160.

[24] H. M. Bücker, B. Lang, A. Rasch, C. H. Bischof, Computation of sensitivity information for aircraft design by automatic differentiation, in: P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, A. G. Hoekstra (Eds.), Computational Science – ICCS 2002, Proceedings of the International Conference on Computational Science, Amsterdam, The Netherlands, April 21–24, 2002. Part II, Vol. 2330 of Lecture Notes in Computer Science, Springer, Berlin, 2002, pp. 1069–1076.

[25] C. H. Bischof, H. M. Bücker, B. Lang, A. Rasch, E. Slusanschi, Efficient and accurate derivatives for a software process chain in airfoil shape optimization, Tech. Rep. RWTH-CS-SC-02-06, Institute for Scientific Computing, Aachen University of Technology, Aachen (2002).

[26] B. Mohammadi, J.-M. Malé, N. Rostaing-Schmidt, Automatic differentiation in direct and reverse modes: Application to optimum shapes design in fluid mechanics, in: M. Berz, C. H. Bischof, G. F. Corliss, A. Griewank (Eds.), Computational Differentiation: Techniques, Applications, and Tools, SIAM, Philadelphia, Penn., 1996, pp. 309–318.

[27] S. Ulbrich, Optimal Control of Nonlinear Hyperbolic Conservation Laws with Source Terms , Habilitationsschrift, Fakultät für Mathematik, Technische Universität München, Germany, 2002.
URL `http://www-m1.mathematik.tu-muenchen.de/m1/personen/sulbrich`

[28] M. A. Park, L. L. Green, R. C. Montgomery, D. L. Raney, Determination of Stability and Control Derivatives Using Computational Fluid Dynamics and Automatic Differentiation, IAAA Paper 1999-3136, AIAA, Reston Va, USA (1999).

[29] A. C. Taylor III, L. L. Green, P. A. Newman, M. M. Putko, Some Advanced Concepts in Discrete Aerodynamic Sensitivity Analysis, IAAA Paper 2001-2529, AIAA, Reston Va, USA (2001).

[30] S. S. Collis, K. Ghayour, M. Heinkenschloss, M. Ulbrich, S. Ulbrich, Towards Adjoint-Based Methods for Aeroacoustic Control, IAAA Paper 2001-0821, AIAA, Reston Va, USA (2001).
URL `http://www.mems.rice.edu/~collis/papers`

[31] S. S. Collis, K. Ghayour, M. Heinkenschloss, M. Ulbrich, S. Ulbrich, Optimal control of unsteady compressible viscous flows, Int. J. Numer. Meth. Fluids

40 (11) (2002) 1401–1429.
URL http://www.mems.rice.edu/∼collis/papers

[32] M. Hinze, T. Slawig, Adjoint Gradients Compared To Gradients From Algorithmic Differentiation In Instantaneous Control Of The Navier-Stokes Equations, Optimization Methods & Software 18 (3) (2003) 299–315.

[33] K. C. Hall, J. P. Thomas, J. P. Clark, Computation of unsteady nonlinear flows in cascades using a harmonic balance technique, AIAA Journal 40 (5) (2002) 879–886.

[34] J. P. Thomas, E. H. Dowell, K. C. Hall, Nonlinear inviscid aerodynamic effects on transonic divergence, flutter and limit cycle oscillations, AIAA Journal 40 (4) (2002) 638–646.

[35] B. Mohamadi, Fluid Dynamics Computation with NSC2KE, User-Guide, Release 1.0, Tech. Rep. RT-0164, INRIA (May 1994).

[36] P. Heimbach and C. Hill and R. Giering, An efficient exact adjoint of the parallel MIT general circulation model, generated via automatic differentiation, to appear in Future Generation Computer Systems .
URL http://www.mit.edu/afs/athena.mit.edu/user/h/e/heimbach/www

[37] J. Marshall, A. Adcroft, C. Hill, L. Perelman, C. Heisey, A Finite-Volume, Incompressible Navier Stokes Model for Studies of the ocean on Parallel Computers, Technical Report 36, Massachusetts Institut of Technology, Center for Global Change Science, Cambridge, MA 02139, USA (1995).

[38] R. Giering, T. Kaminski, Applying TAF to generate efficient derivative code of Fortran 77-95 programs, PAMM 2 (1) (2003) 54–57.
URL
http://www3.interscience.wiley.com/cgi-bin/issuetoc?ID=104084257

[39] R. Giering, T. Kaminski, Generating recomputations in reverse mode AD, in: G. Corliss, A. Griewank, C. Fauré, L. Hascoet, U. Naumann (Eds.), Automatic Differentiation of Algorithms: From Simulation to Optimization, Springer Verlag, Heidelberg, 2002, Ch. 33, pp. 283–291.
URL
http://www.springer.de/cgi-bin/search_book.pl?isbn=0-387-95305-1

[40] R. Giering, T. Kaminski, Recipes for adjoint code construction, ACM Trans. Math. Softw. 24 (4) (1998) 437–474.

[41] B. Christianson, Reverse accumulation and attractive fixed points, Optimization Methods and Software 3 (1994) 311–326.

[42] B. Christianson, Reverse accumulation and implicit functions, Optimization Methods and Software 9 (4) (1998) 307–322.

[43] M. Hinze, Optimal and instantaneous control of the instationary Navier-Stokes equations, Habilitationsschrift, Fachbereich Mathematik, Technische Universität Berlin, 1999.

[44] P. Heimbach, C. Hill, R. Giering, Automatic generation of efficient adjoint code for a parallel Navier-Stokes solver, in: P. M. A. Sloot, C. J. K. Tan, J. J. Dongarra, A. G. Hoekstra (Eds.), Computational Science – ICCS 2002, Proceedings of the International Conference on Computational Science, Amsterdam, The Netherlands, April 21–24, 2002. Part II, Vol. 2330 of Lecture Notes in Computer Science, Springer, Berlin, 2002, pp. 1019–1028.
URL http://www.mit.edu/afs/athena.mit.edu/user/h/e/heimbach/www

[45] S.-J. Lin, R. B. Rood, Multidimensional flux-form semi-lagrangian transport scheme, Mon. Wea. Rev. 124 (9) (1996) 2046–2070.

[46] S.-J. Lin, R. B. Rood, An explicit flux-form semi-lagrangian shallow-water model on the sphere, Quart. J. Roy. Meteor. Soc. 123 (1997) 2477–2498.

[47] S.-J. Lin, A finite volume integration method for computing pressure gradient force in general vertical coordinates, Quart. J. Roy. Meteor. Soc. 123 (1997) 1749–1762.

[48] W. Knorr, Annual and interannual CO_2 exchanges of the terrestrial biosphere: process based simulations and uncertainties, Glob. Ecol. and Biogeogr. 9 (2000) 225–252.

[49] P. Rayner, W. Knorr, M. Scholze, R. Giering, T. Kaminski, M. Heimann, C. L. Quere, Inferring terrestrial biosphere carbon fluxes from combined inversions of atmospheric transport and process-based terrestrial ecosystem models, in: Proceedings of 6th Carbon dioxide conference at Sendai, 2001, pp. 1015–1017.

[50] R. C. Pacanowski, S. M. Griffies, MOM 3.0 Manual, Tech. rep., NOAA/Geophysical Fluid Dynamics Laboratory (1999).
URL
http://www.gfdl.noaa.gov/∼smg/MOM/web/guide_parent/guide_parent.html

[51] E. Galanti, E. Tziperman, M. Harrison, A. Rosati, R. Giering, Z. Sirkes, The equatorial thermocline outcropping - a seasonal control on the tropical pacific ocean-atmosphere instability, Journal of Climate 15 (19) (2002) 2721–2739.

[52] J. Marotzke, R. Giering, Q. K. Zhang, D. Stammer, C. N. Hill, T. Lee, Construction of the adjoint MIT ocean general circulation model and application to atlantic heat transport sensitivity, J. Geophys. Res. 104 (1999) 29,529 – 29,548.
URL
http://jupiter.agu.org/epubs/jgr_oceans/jc9912/1999JC900236/0.html

[53] M. Bücker, R. Beucker, C. Bischof, Using Automatic Differentiation for the Minimal p-Norm solution of the Biogmagnetic Inverse Problem, in: A. Heemink, L. Dekker, H. de Swaan, I. Smit, T. von Stijn (Eds.), Shaping Future with Simulation, Proceedings of the 4-th International Eurosim 2001 Congress, Delft, The Netherlands, June 26–29, 2001, Dutch Benelux Simulation Society, 2001.

[54] H. M. Bücker, R. Beucker, Using automatic differentiation for the solution of the minimum p-norm estimation problem in magnetoencephalography, Simulation Modelling Practice and Theory 12 (2004) 105–116.

[55] M. Heimann, The global atmospheric tracer model TM2, Technical Report No. 10, Max-Planck-Institut für Meteorologie, Hamburg, Germany (1995).

[56] T. Kaminski, M. Heimann, R. Giering, A coarse grid three dimensional global inverse model of the atmospheric transport, 1, Adjoint model and Jacobian matrix, J. Geophys. Res. 104 (D15) (1999) 18,535–18,553.

[57] A. Griewank, Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation, Optimization Methods and Software 1 (1992) 35–54.

[58] R. Giering, T. Kaminski, On the performance of derivative code generated by TAMC, Manuscript, FastOpt, Hamburg, Germany, submitted to Optimization Methods and Software. See `www.FastOpt.de/papers/perftamc.ps.gz`. (2000).