Efficiency of an Adjoint CFD Code for Shape Optimization Problems

Z. Dastouri*
Software and Tools for Computational Engineering (STCE)
RWTH Aachen, LuFG Informatik 12, D-52062 Aachen, Germany
Email: dastouri@stce.rwth-aachen.de

U. Naumann
Software and Tools for Computational Engineering (STCE)
RWTH Aachen, LuFG Informatik 12, D-52062 Aachen, Germany
Email: naumann@stce.rwth-aachen.de

Summary

Sensitivity analysis with the aim of design optimization is a growing area of interest in Computational Fluid Dynamics(CFD) simulations. However, one of the major challenges is to deal with a large number of design variables for large-scale industrial application. One of the effective solution approach is to compute adjoint based sensitivities in the differentiated CFD code. In this paper, we develop a discrete adjoint code for an unstructured pressure-based steady Navier-Stokes flow solver using Algorithmic Differentiation(AD). The objective is to assess the performance of the adjoint code and to improve its efficiency in terms of memory consumption and runtime. To achieve that, we apply the effective techniques to optimize the performance by implementation of check-pointing schemes (Revolve Algorithm[1]) and analytical treatment of the inner iterative linear solver. We combine the flexibility of an operator overloading tool with the efficiency of an adjoint code generated by source transformation. In addition, we improve the performance of adjoint computation by exploiting the mathematical aspect of the involved fixed-point iteration by reverse accumulation.[2] We compare the effectiveness of these methods in terms of numerical cost for the optimization of a vehicle climate duct industrial test case.

Keywords: *Algorithmic Differentiation, Computational Fluid Dynamics Code, Operator Overloading, Discrete Adjoint, Optimization, Sensitivity Analysis*

1 Introduction

Design optimization for fluid flow plays a significant role in wide range of engineering applications including aeronautic, turbomachinery and automotive design. In complex configurations, the optimization problem scales to the large number of design parameters and constraints. The effect of independent design variables on the system performance can be calculated in terms of sensitivities that are the derivatives of one or more quantities (outputs) with respect to one or several independent variables (inputs). By default, these derivatives can be obtained by divided (finite) differences from perturbed solutions. This method is both costly and subject to inaccuracies. Algorithmic Differentiation(AD)[3,4] is a wellknown technique to evaluate derivatives based on the application of the chain rule of differentiation to each operation in the program flow. The derivatives given by the chain rule can be propagated forward (forward mode) or backward (reverse mode). There are two main methods for implementing algorithmic differentiation: by source code transformation (S-T) or by using derived data types and operator overloading (O-O). In O-O AD the code segments and arguments of the primal code are stored inside a memory structure called tape during the forward run of the primal. In reverse mode the stored values on the tape are interpreted to get the resultant derivatives. While in S-T approach the code is parsed at a compile time and the actual code is differentiated.

Prior to this paper, we describe a design framework for application of the AD tool by operator overloading in Fortran *dco/fortran*[1] to CFD analysis solver called GPDE [2]. GPDE is an unstructured pressure-based steady Navier-Stokes solver with finite volume spatial discretization, for the in- compressible viscous flow computation.[6]

[1] Derivative Code by Overloading in Fortran

[2] General Partial Differential Equation solver developed by CFD group of QMUL(http://www.qmul.ac.uk)

In previous paper,[7] we discussed the implementation of dco/fortran in original CFD solver from scratch to get the tangent-linear and adjoint version of the CFD code and also we addressed proper solution algorithms adapted to the code including a equidistant checkpointing scheme for iterative solver and development of a analytical treatment of linear solver inside the code.
In this paper our emphasize is on the improvement of efficiency by replacing the used equidistant checkpointing scheme by REVOLVE.[1] Moreover we combine the flexibility and robustness of operator overloading with the efficiency of source- transformation approach by coupling dco/fortran and TAPENADE[8] for most expensive part of the adjoint code. In addition, we get the benefit of reverse accumulation technique[2] for the fixed point iterative construction in the CFD code that carries out the nonlinear iterations for solving the momentum and mass conservation equations. This implementation shows that the derivatives converge at the same rate of the primal code, displaying of an improvement in performance.

2 CFD-Simulation

The CFD solver we applying for adjoint computation, is an incompressible, steady-state flow solver with cell-centered storage, face-based residual assembly; it works on unstructured meshes with collocated variables, and uses the SIMPLE[9] pressure correction algorithm as a pseudo timestepping scheme towards a steady solution. It is written in Fortran 90-95 (7,000 lines) as a test-bed for developing adjoint Navier-Stokes fields and is specifically designed to facilitate interfacing with optimization libraries.

The case study that is used for flow model simulation and sensitivity studies, is the S-bend channel flow case which is a simplified vehicle climatisation duct. Test case is carried out at Reynold = 500 on hexahedral mesh with 47000 elements. The boundary condition is defined as uniform flow at inlet. The simulation in GPDE is performed for 273 outer iteration in the primal code until we attain the convergence with the tolerance of 9.4413E-08 for velocity reduction. The geometry and velocity vector field along the channel is shown in Figure 1.

3 Effective Adjoint CFD Solver

Simplicity of implementation of the adjoint method by overloading tool comes with a price in taping process. The needed size of this tape grows dynamically proportional to the number of operations in the calculation that leads to considerable memory consumption. The effective techniques need to be implemented to reduce the memory consumption in the adjoint code.

3.1 Binomial Checkpointing

Checkpointing schedules for O-O AD are commonly used in order to reduce the huge memory consumption usually generated by taping a time-dependent iterative solver in plain black box O-O AD. The basic idea is to split the entire program into several sequential blocks of operations whose computational graphs each fit into the available memory. In case of an iterative solver these blocks consist of a certain number of iterations. These blocks are then taped and interpreted one at a time to produce the resulting adjoint values of the complete computation. Unfortunately, this procedure is always a trade-off between memory consumption and runtime due to two reasons:

- The system state at the beginning of a block usually depends on the execution of all previous blocks.
- Blocks have to be taped and interpreted in reverse order compared to their position in the original code.

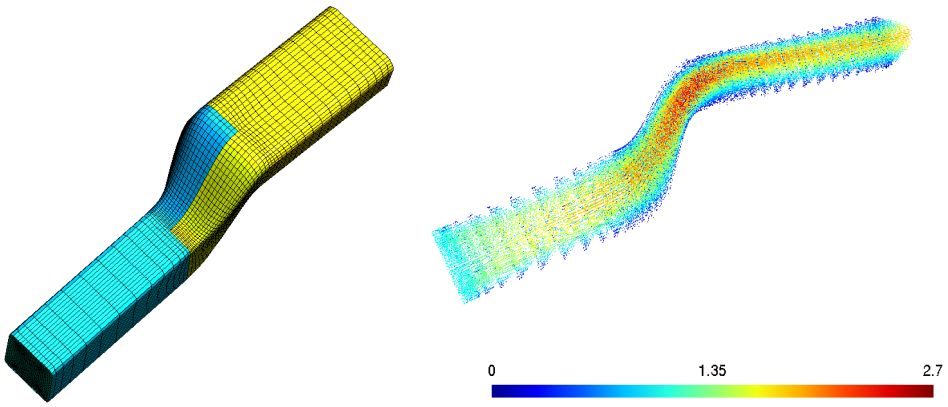
A checkpoint, i.e. saving the system state, is used to avoid recalculating all previous blocks when taping and interpreting blocks in reverse order. The revolve algorithm,[1] introduced by Griewank et al, provides an optimal checkpointing schedule for a prescribed number of n_{cp} checkpoint slots. It always tapes only one time step and minimizes the amount of extra forward steps needed by setting checkpoints in a binomial fashion. This method yields logarithmic grows in spatial complexity.

3.2 Reverse Accumulation

The CFD procedure consists of iterative loops for flow equations. Hence the memory requirement of the reverse mode grows proportionally with the number of iterations. The fixed point iterative loop in CFD code carries out outer iterations for solving the momentum and mass conservation equations. Therefore, the reverse accumulation technique[2] for fixed point iterative construction in the CFD code is implemented. By applying this technique we reduce memory consumption of the adjoint code independent of the number of iterations. The implementation of the method using dco/fortran O-O tool is illustrated in Algorithm 1 and figure 2.

3.3 Hybrid overloading-source transformation approaches

The CFD code is alternatively differentiated by source transformation tool, TAPENADE.[2] TAPENADE is an Automatic Differentiation tool which, given a Fortran or C code that computes a function, creates a new code that computes its tangent or adjoint derivatives. This approach reduces the memory requirements for the differentiating purposes and it is easier for the compiler to do compile time optimizations. However in terms of ease of implementation and ability to handle arbitrary functions, operator overloading provides the differentiated code with a greater flexibility and robustness in comparison with AD by source transformation. Therefore coupling these two AD tools remains an efficient approach to decrease in one hand the development time of differentiated code and in other hand to reduce the memory requirement of the adjoint code. The implementation steps for applying TAPENADE AD tool via a dco tape is explained in Figure 3 and Algorithm 2. dco/fortran provides an interface to use external functions



(a) S-Bend vehicle climatisation duct geometry (b) Velocity vector field though the geometry center

Figure 1: Geometry and velocity vector field of the S-bend channel flow

Calling the driver code
Call dco_get_tape_position(pos1)
Call switch_tape_to_passive()
do $i = 1, n\_iter$
if $i = n-1$ **then**
  Call switch_tape_to_active()
  Call dco_get_tape_position(pos2)
else
  Call $flow(U,P,n\_iter)$
end
end do
Call dco_get_tape_position(pos3)
Call $objective(U^n, P^n, obj, n\_iter)$
Call dco_a1s_set($obj_{(1)} = 1$)
call dco_a1s_tape_interpret_adjoint_to(pos3) Call dco_a1s_get($(U_{r(1)}, P_{r(1)}) = (U^n_{(1)}, P^n_{(1)})$)
Call dco_a1s_set($U^0_{(1)}, P^0_{(1)}) = 0$
while $i < n.and.\xi < \varepsilon$ **do**
  Call dco_a1s_tape_zero_adjoints
  Call dco_a1s_set($U^i_{(1)}, P^i_{(1)}) =$
  $(U^{i-1}_{(1)} + U_{r(1)}, P^{i-1}_{(1)} + P_{r(1)})$
  call
  dco_a1s_tape_interpret_adjoint_from_to(pos3,pos2)
  Call dco_a1s_get($(U^i_{(1)}, P^i_{(1)})$)
end
call dco_a1s_tape_interpret_adjoint_from(pos1)

Algorithm 1: Implementation of reverse accumulation using dco/fortran

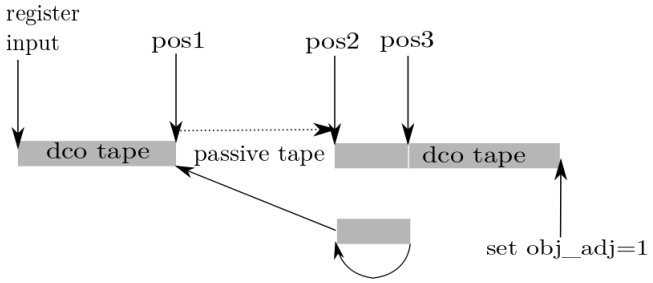


Figure 2: Reverse accumulation during the taping process

for the calculation of the adjoint. Instead of recording the operations of such a function inside the forward-run, this function is registered in the tape to call and execute in reverse run of the adjoint code. The idea is to extract the computationally most expensive part of the adjoint code and differentiate it by TAPENADE engine. The adjoints generated by TAPENADE increment back to dco tape.

4 Numerical Cost Comparison

Our approach in this paper is adjoint based shape optimization problem from development the adjoint CFD code to design optimization. We overview and implement efficient techniques for improving the performance of the adjoint CFD code and we demonstrate significant improvement in terms of memory consumption and runtime of the resulting code for a sample CFD problem. Sensitivity analysis results are presented for pressure loss objective function with respect to surface boundary nodes of S-bend in three dimensions. We apply the sensitivity analysis results to optimize the shape of the surface in the flow

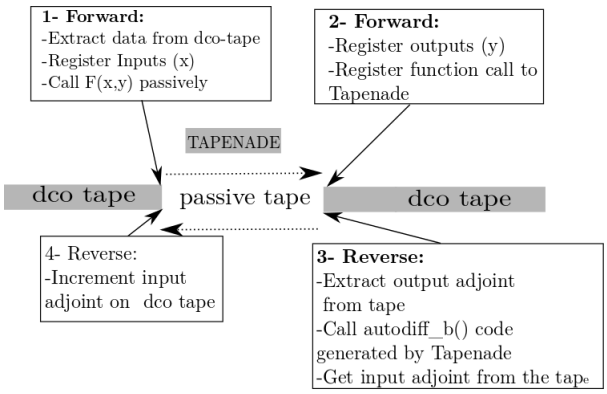


Figure 3: Calling TAPENADE from dco Tape

Forward Sweep
Call dco_a1s_checkpoint_create(cp)
Call dco_a1s_register_input(cp,U)
Call dco_a1s_register_input(cp,gradU)
Call dco_a1s_register_input(cp,X)
Call switch_tape_to_passive()
Call gradient_passive(U,X,gradU,n_iter)
Call switch_tape_to_active()
Call dco_a1s_register_output(cp,U)
Call dco_a1s_register_output(cp,gradU)
Call dco_a1s_register_output(cp,X)
Call
dco_a1s_register_external_function(gradient_ext,cp)
Reverse Sweep
Call
dco_a1s_read_incoming_adjoint_from_tape(geom_b, U_b, grad_b)
!Specify diffhead TAPENADE Call grad_b(geom, geom_b, U, U_b, grad, grad_b, n_iter)
Call
dco_a1s_write_outgoing_adjoint_to_tape(geom_b, U_b, grad_b)

Algorithm 2: Implementation steps-Coupling dco and Tapenade

domain.

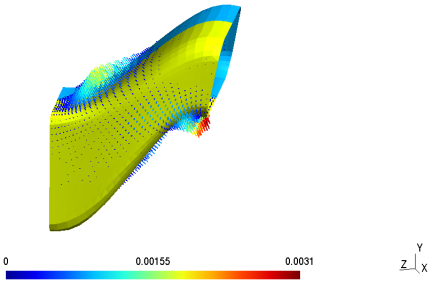


Figure 4: Surface sensitivity results S-Bend

ACKNOWLEDGEMENT

The presented research is supported by the project **aboutFlow**, funded by the European Commission under FP7-PEOPLE-2012-ITN-317006.

References

[1] Griewank, A. and Walther, A. Revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation. Algorithm 799: ACM Transactions on Mathematical Software, 26(1):1945, March 2000.

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

[3] The Art of Differentiating Computer Programs. An Introduction to Algorithmic Differentiation. Software, Environments, and Tools. SIAM, Philadelphia, PA, 2012.

[4] Griewank, A. Walther, A. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, second edition. SIAM: Philadelphia, PA, 2008.

[5] Naumann, U. AD-enabled NAG Fortran compiler, dco/fortran:User Guide, Software and Tools for Computational Engineering institute, RWTH Aache, 2013

[6] Jones D, Christakopoulos F, Müller J-D. Preparation and assembly of adjoint CFD codes. Computers and Fluids 2011; 46(1):282–286. London, 2012.

[7] Dastouri, Z., Lotz, J., Naumann, U. Development of a Discrete Adjoint CFD Code using Algorithmic Differentiation by Operator Overloading. OPTi2014. June 4-6, 2014, Kos, Greece.

[8] Hascoët, L., Pascual, V. The Tapenade automatic differentiation tool: Principles, model, and specification. ACM, Transactions On Mathematical Software, Volume 39 Issue 3, Article No. 20, April 2013.

[9] Patankar,S.V. , SpaldingD.B. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. International Journal for Heat Mass Transfer, Number 15:1787-1806, 1972.