Computer Physics Communications 199 (2016) 22–28

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

The efficiency of geophysical adjoint codes generated by automatic differentiation tools

A.V. Vlasenko [*], A. Köhl, D. Stammer

Center für Erdsystemforschung und Nachhaltigkeit, Universität Hamburg, Germany

ARTICLE INFO

Article history:
Received 14 January 2015
Received in revised form
4 October 2015
Accepted 18 October 2015
Available online 30 October 2015

Keywords:
Sensitivity estimation
Computational efficiency
Data assimilation

ABSTRACT

The accuracy of numerical models that describe complex physical or chemical processes depends on the choice of model parameters. Estimating an optimal set of parameters by optimization algorithms requires knowledge of the sensitivity of the process of interest to model parameters. Typically the sensitivity computation involves differentiation of the model, which can be performed by applying algorithmic differentiation (AD) tools to the underlying numerical code. However, existing AD tools differ substantially in design, legibility and computational efficiency. In this study we show that, for geophysical data assimilation problems of varying complexity, the performance of adjoint codes generated by the existing AD tools (i) Open_AD, (ii) Tapenade, (iii) NAGWare and (iv) Transformation of Algorithms in Fortran (TAF) can be vastly different. Based on simple test problems, we evaluate the efficiency of each AD tool with respect to computational speed, accuracy of the adjoint, the efficiency of memory usage, and the capability of each AD tool to handle modern FORTRAN 90–95 elements such as structures and pointers, which are new elements that either combine groups of variables or provide aliases to memory addresses, respectively. We show that, while operator overloading tools are the only ones suitable for modern codes written in object-oriented programming languages, their computational efficiency lags behind source transformation by orders of magnitude, rendering the application of these modern tools to practical assimilation problems prohibitive. In contrast, the application of source transformation tools appears to be the most efficient choice, allowing handling even large geophysical data assimilation problems. However, they can only be applied to numerical models written in earlier generations of programming languages. Our study indicates that applying existing AD tools to realistic geophysical problems faces limitations that urgently need to be solved to allow the continuous use of AD tools for solving geophysical problems on modern computer architectures.

1. Introduction

To date, numerical modeling is a widespread and generally accepted approach for solving complex mathematical equations of physical, biological, or chemical processes in climate and earth system sciences. However, the accuracy of respective solutions fundamentally depends on the choice of – typically uncertain – model parameters. One therefore is usually faced with the following two questions: (i) How sensitive are model solutions to the detailed choice of model parameters, and (ii) what is the optimal set of these parameters required to minimize the difference between a simulated process to a given set of observations. One elegant way to answer these questions involves the computation of derivatives of the corresponding numerical model with respect to its parameters or state variables [1–5].

Generally, model derivatives can be generated in three different ways, the simplest of which is an estimation of approximate derivatives by applying a finite difference method. Being simple, however, this method is always plagued by approximation errors. The second way is to derive the model derivatives manually. Such differentiation leads to exact derivatives but is labor-intensive and therefore impractical for large numerical models. The third option is to use an algorithmic differentiation (AD) tool, which almost automatically provides the exact derivatives of any complex function represented by a numerical code with only little extra effort by the user.

For simplicity we introduce two definitions commonly used in AD. We consider a numerical model as mathematical operator that acts on state variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and returns an output $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ (e.g. state variables at the end of computation). Let $\mathbf{J}$ be the Jacobian matrix of $\boldsymbol{y}$ with respect to $\boldsymbol{x}$ and $\boldsymbol{s}$ be

* Corresponding author.
E-mail address: andrey.vlasenko@uni-hamburg.de (A.V. Vlasenko).

http://dx.doi.org/10.1016/j.cpc.2015.10.008
0010-4655/© 2015 Elsevier B.V. All rights reserved.

an n-dimensional column vector, with only s_i element equal to one and the rest to zero. The differentiation of $\mathbf{y}$ with respect to x_i produced by the AD tool can be presented as a multiplication of $\boldsymbol{s}$ by $\mathbf{J}$, where $\mathbf{J}\boldsymbol{s}$ gives the derivative of $\boldsymbol{y}$ with respect to x_i. The corresponding computer code, generated by the AD tool, is called the tangent linear model and the derivatives referred to as tangents. The multiplication of $\boldsymbol{s}^{\boldsymbol{T}}\mathbf{J}$ (superscript T means transpose) provides the derivative of y_i with respect to $\boldsymbol{x}$. The corresponding computer code, generated by the AD tool, is called the adjoint model and the derivatives referred as adjoints. These two ways of differentiation of a numerical model performed by the AD tool are referred as tangent linear and adjoint modes,[1] respectively. A further field of AD tools is the computation of higher order derivatives. In particular, the second derivative, the Hessian matrix, or approximations to it, are used in parameter optimization algorithms such as the Gauss–Newton algorithm, and the posterior evaluation of the uncertainties of the estimated parameters requires the inverse of the Hessian.

Today, AD is a well-established field of applied mathematics, formulated initially in the middle of the previous century. In detail, AD techniques are based on the mathematical formalisms of differentiation represented by a set of simple, well-known mathematical operations. They are designed for the numerical differentiation of mathematical functions of any complexity represented by computer codes [6,7]. The resultant derivatives are exact up to the numerical precision, i.e., no truncation is used so that no approximation errors appear in the output code.

By now more than 42 different AD packages exist, which were developed for different applications and programming languages (see www.autodiff.org for details). They use different strategies for computing derivatives [8]. Among them source code transformation (SCT) and operator overloading (OO) are the most common strategies [9]. SCT AD tools replace the original source code by a new source code, usually in the same programming language that includes the statements for computing user specified derivatives. In contrast, the OO AD tools generally leave the original source code as it is and the sequence of operations in the original source code for the function to be differentiated remains unchanged. However, it requires a change in basic data types for numbers and vectors for supporting differentiation, and establishes a special polymorphism[2] for mathematical operators called overloading. The change of data type is done for all differentiating and differentiated variables (which may be done automatically by the AD tool) in such a way, that any variable of a changed type holds both its actual value and its derivative. Overloading reintroduces the original mathematical operators in the source code by splitting each operator's action. An action of any overloaded elementary mathematical operator depends on whether it is applied to the value of the variable or to its derivative. For an actual value of the variable, the operator remains the same as it was before the overloading, while it operates according to the rules of differentiation when being applied to the derivatives of the variable.

Respective tools have been applied to a variety of problems in many research areas [10–17]. But, although all these AD tools simplify the computation of the derivatives, the computational efficiency of these derivatives strongly depends on the internal structure of the AD tool that produced it. In particular, existing AD tools differ substantially in the design, readability and computational efficiency of resulting numerical codes. Moreover, the AD tools also differ by their ability to operate with various high level programming languages. This difference in structure and design of different AD tools has a great effect on the performance of the adjoints that they produce. Applying these theories to real geophysical problems remains a huge challenge given the size, complexity, and often the non-linear nature of those problems. Understanding the efficiency of existing adjoint codes is therefore a prerequisite to applying them to real climate science problems. Rather than developing new algorithms, the goal of this paper is to analyze the practicality and efficiency of existing OO and STC based AD tools for geophysical problems. We focus our study on adjoint modes, since the computation of sensitivities in climate modeling is mainly associated with the execution of adjoint models. Based on simple test problems, we evaluate the efficiency of each AD tool with respect to computational speed, accuracy of the generated adjoint, the efficiency of memory usage, and the capability of each AD tool to handle modern FORTRAN 90–95 elements such as structures and pointers.

In detail, we compare adjoint codes generated by Transformation of Algorithms in Fortran (TAF) [18] and Open_AD [19] and TAPENADE [20] as the AD tools representing SCT approach. As for the OO based AD tool, we choose NAGWare [21]. All these AD tools have the tangent linear and adjoint mode for differentiation. We note that the same derivatives can be obtained with either an adjoint or a tangent linear mode; which mode is more appropriate is a question of efficiency.

Here, we focus on the performance of the adjoint mode of differentiation, because most of difficulties related to memory usage efficiency and execution runtime are typically associated with this mode [22]. Our case studies are specifically intended to identify strengths and weaknesses of the OO and SCT procedures and provide the corresponding benchmarks for their efficiency in climate applications. The ability to operate with different programming language features is also compared, and their effect on the computational speed performance (CSP) is investigated. By doing so, we expect to give recommendations to model developers guiding their choice of the proper AD procedures.

The remaining paper is organized as follows. A brief summary of the methodology for testing the compilers is given in Section 2. The test-bed codes on which the efficiency of the compilers is investigated is given in Section 3. Section 4 gives a short theoretical overview of the differentiation machinery of the AD tools. Tests on the compiler's speed performance, accuracy and the efficiency in memory usage are described in fifth and six sections respectively. Section 7 describes the ability of the compilers to handle pointers and user defined structures. The conclusion is given in Section 8.

2. Test of the AD tools

Our inter-comparison is based on the latest versions of TAF (versions 2.3.5-2.8.4), Open_AD (versions S440, S469, S493), Tapenade (version 3.10) and NAGWare (version 5.1) AD packages. For a quantitative comparison of all three AD tools, we have prepared several sample codes to test computational speed, memory usage efficiency (MUE), accuracy, and compatibility with different programming features such as pointers, structures, and user defined data types. In order to test how the efficiency of computations of adjoints depends on the hardware, the tests were carried out on different platforms. All executables were subsequently run on three different computer platforms (Intel CoreDuo E8500 2 Gb DDR3, AMD Sempron AM2 2 Gb DDR2 and Intel Core I5-2500 8 Gb DDR3), and the corresponding computational times were used to evaluate the performance. Although the SCT AD tools allow utilizing any FORTRAN compiler, the NAGWare AD tool is embedded in the NAG FORTRAN compiler, and consequently compilation has to be done with this. To avoid differences in execution time related to the usage of different

[1] The terms adjoint and tangent linear **models** should not be confused with terms adjoint and tangent linear **modes**.

[2] Polymorphism is the ability of a function to be applied to different types of variables. The result of application depends on the types of these variables.