Computers and Fluids 168 (2018) 285–303

Contents lists available at ScienceDirect

Computers and Fluids

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

An aerodynamic design optimization framework using a discrete adjoint approach with OpenFOAM

Ping He[a,*], Charles A. Mader[a], Joaquim R.R.A. Martins[a], Kevin J. Maki[b]

[a] Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan 48109, USA
[b] Department of Naval Architecture and Marine Engineering, University of Michigan, Ann Arbor, Michigan 48109, USA

ARTICLE INFO

Article history:
Received 8 November 2017
Revised 9 March 2018
Accepted 6 April 2018
Available online 7 April 2018

Keywords:
OpenFOAM
Discrete adjoint optimization
Parallel graph coloring
Ahmed body
UAV
Car

ABSTRACT

Advances in computing power have enabled computational fluid dynamics (CFD) to become a crucial tool in aerodynamic design. To facilitate CFD-based design, the combination of gradient-based optimization and the adjoint method for computing derivatives can be used to optimize designs with respect to a large number of design variables. Open field operation and manipulation (OpenFOAM) is an open source CFD package that is becoming increasingly popular, but it currently lacks an efficient infrastructure for constrained design optimization. To address this problem, we develop an optimization framework that consists of an efficient discrete adjoint implementation for computing derivatives and a Python interface to multiple numerical optimization packages. Our adjoint optimization framework has the following salient features: (1) The adjoint computation is efficient, with a computational cost that is similar to that of the primal flow solver and scales up to 10 million cells and 1024 CPU cores. (2) The adjoint derivatives are fully consistent with those generated by the flow solver with an average error of less than 0.1%. (3) The adjoint framework can handle optimization problems with more than 100 design variables and various geometric and physical constraints such as volume, thickness, curvature, and lift constraints. (4) The framework includes additional modules that are essential for successful design optimization: a geometry-parametrization module, a mesh-deformation algorithm, and an interface to numerical optimizations. To demonstrate our design-optimization framework, we optimize the ramp shape of a simple bluff geometry and analyze the flow in detail. We achieve 9.4% drag reduction, which is validated by wind tunnel experiments. Furthermore, we apply the framework to solve two more complex aerodynamic-shape-optimization applications: an unmanned aerial vehicle, and a car. For these two cases, the drag is reduced by 5.6% and 12.1%, respectively, which demonstrates that the proposed optimization framework functions as desired. Given these validated improvements, the developed techniques have the potential to be a useful tool in a wide range of engineering design applications, such as aircraft, cars, ships, and turbomachinery.

1. Introduction

Open field operation and manipulation (OpenFOAM) is an open source software package for computational fluid dynamics (CFD) [1,2] that contains more than 80 solvers capable of simulating various types of flow processes, including aerodynamics, hydrodynamics, heat transfer, and multiphase flow [3]. OpenFOAM is being actively developed and verified by its users and developers [4–7], and its popularity has been rapidly growing over the past decade. OpenFOAM has become a powerful tool for aerodynamic design of engineering systems such as aircraft, cars, and turbomachinery [8–14]. One of the major tasks in the process of aerodynamic design is to improve system performance (e.g., reduce drag, maximize power, and improve efficiency). Traditionally, it involves manual loops of design modification and performance evaluation, which is not efficient. To improve the efficiency of this process, the combination of gradient-based optimization and the adjoint method for computing derivatives can be used to automatically optimize the design. The true benefit of using the adjoint method to compute derivatives is that its computational cost is almost independent of the number of design variables, which enables complex industrial design optimization. Given this background, the development of an adjoint optimization framework may facilitate the existing process of OpenFOAM-based aerodynamic-shape design.

The adjoint method was first introduced to fluid mechanics by Pironneau [15] in 1970s. The approach was then extended by

* Corresponding author.
E-mail address: drpinghe@umich.edu (P. He).

https://doi.org/10.1016/j.compfluid.2018.04.012
0045-7930/© 2018 Elsevier Ltd. All rights reserved.

Jameson [16] to the optimization of two-dimensional aerodynamic-shape design in the late 1980s. Since then, the adjoint method has been implemented for three-dimensional turbulent flows, and its application has also been generalized to multipoint and multidisciplinary design optimization [17–26]. While the adjoint method is recognized as an efficient method for computing derivatives of a solver based on partial differential equations (PDEs), successful optimization requires a framework that includes other components that go beyond the flow solution and derivative computation. We also require modules for geometry manipulation, mesh deformation, and optimization algorithms. The speed and accuracy of such modules, especially as they pertain to derivative computation, strongly impact the overall optimization. We have developed a full suite of modules to facilitate aerodynamic optimization, some of which have been published previously. The geometry-manipulation module was developed by Kenway et al. [27] and has been used in various aerodynamic and aerostructural design-optimization studies [26,28–31]. Perez et al. [32] developed an open source Python interface to various numerical optimization packages that we reuse here.[1] In the present work, we focus on the implementation of the adjoint solver in OpenFOAM, the development of which allows the OpenFOAM solver to be efficiently integrated into our existing optimization framework.

Two different methods exist for formulating the adjoint of a flow solver: continuous and discrete [33]. The continuous approach derives the adjoint formulation from the Navier–Stokes (NS) equations and then discretizes to obtain the numerical solution. In contrast, the discrete approach starts from the discretized NS equations and differentiates the discretized equations to get the adjoint terms. Although these two approaches handle adjoint formulation in different ways, they both converge to the same answer for a sufficiently refined mesh [34].

The adjoint method was first implemented in OpenFOAM by Othmer [35], who used the continuous approach to derive the adjoint formulation for the incompressible flow solver simpleFoam. This continuous adjoint implementation was then integrated as a built-in OpenFOAM solver for computing derivatives. A number of recent studies have reported shape optimization based on derivatives computed from the continuous adjoint [36–40]. Othmer's continuous adjoint framework uses a free-form deformation (FFD) geometry-morphing technique that can handle complex geometries such as full-scale cars. Moreover, the computational cost for the adjoint is similar to that for the primal flow solver, allowing one to tackle cases with more than 10 million cells [38,39]. However, they used a basic steepest descent optimization algorithm to update the shape, so their optimization problems did not include design constraints.

More recently, Towara and Naumann [41] reported a discrete adjoint implementation for OpenFOAM. They used reverse mode automatic differentiation (AD) to compute derivatives so that the adjoint derivatives are fully consistent with the flow solution, regardless of the mesh refinement. However, they used AD to differentiate the entire OpenFOAM code, requiring all flow variables to be stored to conduct the reverse AD computation. To reduce the memory required to store the flow variables, the checkpointing technique was used to trade speed for memory. As a result, the overall computational cost to compute derivatives is high—the adjoint-flow runtime ratio ranges from 5 to 15 [42–44]. Given the cost of this adjoint computation, it would be hard to use this implementation for practical shape optimization.

Instead of applying AD to the entire code, we implement a discrete adjoint approach where the partial derivatives in the adjoint equations are computed by using finite differences (see Section 2.5). The objective here is to develop an adjoint solver within the limitations of the OpenFOAM framework that is sufficiently efficient for practical shape optimization. We evaluate the performance of our adjoint implementation in terms of speed, scalability, and accuracy, optimize the aerodynamic shape of a bluff geometry representative of a ground vehicle, and validate the optimized result by comparing it with the result of wind tunnel experiments. Furthermore, we demonstrate the constrained optimization capability for two more complex shape-optimization applications: an unmanned aerial vehicle (UAV), and a car. We opt to use the discrete adjoint approach because the adjoint derivative is consistent with the flow solution, as mentioned above. Moreover, we find the discrete adjoint implementation easier to maintain and extend (for example, when adding new objective or constraint functions and boundary conditions).

The rest of the paper is organized as follows: Section 2 introduces the optimization framework along with the theoretical background for each of its modules. Section 3 evaluates its performance and presents the aerodynamic shape optimization results. Finally, we summarize and give conclusions in Section 4.

2. Methodology

The design-optimization framework implements a discrete adjoint for computing the total derivative $\mathrm{d}f/\mathrm{d}\boldsymbol{x}$, where f is the function of interest (which for optimization will be the objective and constraint functions, e.g., drag, lift, and pitching moment), and $\boldsymbol{x}$ represents the design variables that control the geometric shape via FFD control point movements. The design-optimization framework consists of multiple components written in C++ and Python and depends on the following external libraries and modules: OpenFOAM, portable, extensible toolkit for scientific computation (PETSc) [45,46], pyGeo [27], pyWarp [27], and pyOptSparse [32]. The framework also requires an external optimization package, which can be any package supported by the pyOptSparse optimization interface. In this section, we elaborate on the overall adjoint optimization framework, the theoretical background for the framework modules, and the code structure and implementation.

2.1. Discrete adjoint optimization framework

Fig. 1 shows the modules and data flow for the optimization framework. We use the extended design structure matrix standard developed by Lambe and Martins [47]. The diagonal entries are the modules in the optimization process, whereas the off-diagonal entries are the data. Each module takes data input from the vertical direction and outputs data in the horizontal direction. The thick gray lines and thin black lines denote the data and process flow, respectively. The numbers in the entries are their execution order.

The framework consists of two major layers: OpenFOAM and Python, and they interact through input and output files. The OpenFOAM layer consists of a flow solver (simpleFoam), an adjoint solver (discreteAdjointSolver), and a graph-coloring solver (coloringSolver). The flow solver is based on the standard OpenFOAM solver simpleFoam for steady incompressible turbulent flow. The adjoint solver computes the total derivative $\mathrm{d}f/\mathrm{d}\boldsymbol{x}$ based on the flow solution generated by simpleFoam. The mesh deformation derivative matrix ($\mathrm{d}\boldsymbol{x_v}/\mathrm{d}\boldsymbol{x}$, where $\boldsymbol{x_v}$ contains the volume-mesh coordinates) is needed when computing the total derivative and is provided by the Python layer. To accelerate computation of the partial derivatives, we developed a parallel graph-coloring solver, whose algorithm is discussed in Section 2.6.

The Python layer is a high-level interface that takes the user input and the total derivatives computed by the OpenFOAM layer and calls multiple external modules to perform constrained optimization. To be more specific, these external modules include "py-

[1] https://www.github.com/mdolab/pyoptsparse.