Journal of Computational Physics 471 (2022) 111664

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Parallel-in-time adjoint-based optimization – application to unsteady incompressible flows

S. Costanzo[a,d,*], T. Sayadi[a], M. Fosas de Pando[b], P.J. Schmid[c], P. Frey[d]

[a] *Sorbonne Université, CNRS, Institut Jean Le Rond d'Alembert, F-75005 Paris, France*
[b] *Department of Mechanical Engineering and Industrial Design, Universidad de Cádiz, Cadiz, Spain*
[c] *Department of Mechanical Engineering, KAUST, 23955 Thuwal, Saudi Arabia*
[d] *Sorbonne Université, Institut des Sciences du Calcul et des Données, ISCD, F-75005 Paris, France*

ARTICLE INFO

Article history:
Received 15 December 2021
Received in revised form 3 September 2022
Accepted 27 September 2022
Available online 6 October 2022

Keywords:
Flow control
Adjoint-based methods
Parallel-in-time algorithm

ABSTRACT

Gradient-based optimization algorithms, where gradient information is extracted using adjoint equations, are efficient but can quickly slow down when applied to unsteady and nonlinear flow problems. This is mainly due to the sequential nature of the algorithm, where the primal problem is first integrated forward in time, providing the initial condition for the adjoint problem, which is then integrated backward. In order to address the sequential nature of this optimization procedure parallel-in-time algorithms can be employed. However, the characteristics of the governing equations of interest in this work, and in particular, the divergence-free constraint (incompressibility effect) as well as the nonlinearity and the unsteadiness of the flow, make direct application of existing parallel-in-time algorithms less than straightforward. In this work, we introduce a parallel-in-time procedure, applied to the integration of the adjoint problem, which addresses all the existing constraints and allows quick access to local gradients. The performance of the proposed algorithm is assessed for both steady and unsteady actuation; in both cases it readily outperforms the sequential algorithm.

1. Introduction

Numerical simulations of multiphysics and multiscale phenomena in fluid mechanics have advanced remarkably over the past decades, and complex physical processes can now be simulated with an astonishing degree of fidelity and accuracy. While it is important to be able to simulate such complex flows with increasing degree of confidence, it is just as crucial to be able to extract relevant optimization and control strategies dedicated to improving performance and efficiency. Although advances in computing capability and software have made computational fluid dynamics (CFD) a valuable tool in determining control strategies in a limited number of applications (including aero-acoustics and shape optimization), as of yet, the application of high-fidelity control strategies to ever more complex systems of equations remains vastly unexplored and resorts systematically to reduced-order models [1,2]. While effective in their own right, these models are unable to describe the full range of dynamics of the system and their potential coupling under various operating conditions. In many applications, a control strategy based on detailed simulations is required. Among the two general classes of optimization algorithms, (i) gradient-based and (ii) derivative-free, the former, when applicable, proves to be the most efficient. It will be the focus of this study. Originally arising as part of a design algorithm for fluid systems [3–6], adjoint methods have been

* Corresponding author at: Sorbonne Université, CNRS, Institut Jean Le Rond d'Alembert, F-75005 Paris, France.
E-mail address: serena.costanzo@dalembert.upmc.fr (S. Costanzo).

https://doi.org/10.1016/j.jcp.2022.111664
0021-9991/© 2022 Elsevier Inc. All rights reserved.

applied to aero- and thermo-acoustic applications [7,8] (dominated by linear dynamics), and to nonlinear systems [9–11]. Gradient-based optimization techniques have also been incorporated into reactive and multi-phase simulations [12–21]. While very efficient and flexible, these algorithms still suffer from a great many challenges.

Recently, challenges encountered in the application of gradient-based techniques to extract control strategies in the presence of turbulence have been tackled, and promising results have been reported in [22–24]. In particular, Vishnampet et al. [25] proposed an exact space-time discrete-adjoint formulation able to predict the gradient at all turbulent scales with reduced cost and effort. This method has been extended to chemically reacting mixtures in the work of Capecelatro et al. [26,16,27,28], allowing the optimization of chaotic configurations, albeit over short time horizons.

On the algorithmic side, on the other hand, a key challenge is associated with the unsteadiness of the flow and the integration of the reverse problem. The evolution of the adjoint variable is governed by a linear, variable-coefficient dynamical system with a general structure similar to the forward problem, except that the adjoint is integrated backward in time. The variable-coefficient nature of the adjoint equations dictates that the solution of the forward integration is needed at each time step of the adjoint problem. This solution must be either stored in memory or recalculated from forward solutions at specifically chosen time instants, referred to as checkpoints. In large scale high-fidelity simulations, relevant in engineering applications, many time steps are usually required for each forward integration, leading to excessive memory requirements to store the solutions. Checkpointing schemes in which only a small number of time steps is stored provide a remedy. In this approach, the solution is stored at carefully chosen checkpoints, and during the backward integration of the adjoint equations, the discarded intermediate solutions are then restored by starting anew the forward integration from the respective checkpoint. Various checkpointing algorithms exist which aim to optimize the number of stored points in memory and the time required for the respective forward integration to access the intermediate solutions [29]. In unsteady cases, the use of checkpointing algorithms increases the computational costs nearly by a factor of three. In addition, the overall time to solution increases proportionally, since these operations (forward and backward integrations) are executed sequentially. While the cost of the calculation cannot be circumvented (adjoint equations need to be solved in order to gain access to derivative information), strategies can be adopted to reduce the overall time to solution, such that, ideally, once the forward problem has reached the final time, the gradient is also available simultaneously. One way of reducing the total time to solution is using parallel-in-time algorithms.

Time-parallel integration has been an active area of research, which started with the pioneering work of Nievergelt [30]. The existing space-time parallel methods can be divided into four main categories: (i) methods based on multiple shooting, (ii) methods based on domain decomposition and waveform relaxation, (iii) methods based on multigrid, and (iv) direct time-parallel methods [31]. While most of these efforts concentrated on accelerating the integration of the direct (forward) simulations, some efforts also considered incorporating a direct-adjoint optimization procedure, such as Maday et al. [32] and Skene et al. [33]. Here, we will concentrate on accelerating the adjoint equations using a parallel-in-time approach introduced by Gander and Güttel [34]. While the applicability of this parallel-in-time methodology to adjoint-based optimization has been analyzed by Skene et al. [33], and multiple possible algorithms have been suggested, the system of equations considered here adds multiple layers of intricacy which have not been addressed in previous studies. One such challenges is due to the complexity of the problem, i.e. the nonlinear unsteady Navier-Stokes equations, compared to Burgers' equation studied previously. However, the main difficulty is due to the algebraic-differential nature of the governing equations (owing to the divergence-free constraint), which makes the application of the exponential time integrator nontrivial and also leads to an algebraic formulation of the adjoint equations. For the parallel-in-time strategy to be relevant for real scale applications, these underlying issues need to be properly addressed, thus motivating the work presented here.

The paper is organized as follows. Section 2 gives an overview of adjoint-based optimization methods, section 4 describes the parallel-in-time procedure. The governing equations and the numerical framework are introduced in section 3. Finally, results of our study are presented in section 5, and conclusions from our work are offered in section 6.

2. Adjoint-based framework

In this section, a methodology for the evaluation of the gradient of a general cost function based on the adjoint equations is presented. In what follows, we consider general algebraic-differential equations, linked to the formulation of the incompressible Navier-Stokes equations. For a detailed classification of various differential equations and their respective adjoint equations see Cao et al. [35].

In the context of control and optimization problems, the variables solved during the forward integration are referred to as the state variables (denoted $\boldsymbol{q}$). In addition to the state variables, a set of parameters, $\boldsymbol{g}$, is also defined as the control or design variables. The cost functional, $\mathcal{J}$, refers to a functional expression based on the state and control variables that is ultimately optimized (i.e. minimized or maximized). Finally, a set of constraints is described by governing equations and side conditions that the state and control variables have to satisfy. Therefore, the resulting optimization problem can be formally stated as follows:

$$\min_{\mathbf{g}} \qquad \mathcal{J}(\boldsymbol{q},\boldsymbol{g}), \quad \text{where} \quad \mathcal{J}(\boldsymbol{q},\boldsymbol{g}) \equiv \int_0^T J(\boldsymbol{q},\boldsymbol{g},t)\,\mathrm{d}t, \tag{1.1}$$

$$\text{s.t.} \qquad \mathbf{F}(\boldsymbol{q},\dot{\boldsymbol{q}},\boldsymbol{g},t) = 0 \ \text{ for } \ 0 \le t \le T, \ \text{ and } \mathbf{G}(\boldsymbol{q}(0),\boldsymbol{g}) = 0, \tag{1.2}$$

where t is the time, T stands for the integration time, $\dot{\boldsymbol{q}}$ denotes the temporal derivative of the state vector, $\mathbf{F}$ represents the nonlinear differential operator, and the function $\mathbf{G}$ has been introduced to allow for initial conditions that depend on the vector of design parameters $\boldsymbol{g}$. In the context of derivative-based optimization, we are interested in computing the gradient of the cost functional with respect to the parameters $\boldsymbol{g}$. Fréchet-differentiating equations (1.1) and (1.2) with respect to $\boldsymbol{g}$, we arrive at

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}\boldsymbol{g}} = \int_0^T \left(\frac{\partial J}{\partial \boldsymbol{q}} \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}\boldsymbol{g}} + \frac{\partial J}{\partial \boldsymbol{g}} \right) \mathrm{d}t, \tag{2.1}$$

where the sensitivity $\mathrm{d}\boldsymbol{q}/\mathrm{d}\boldsymbol{g}$ is determined by

$$\frac{\partial \mathbf{F}}{\partial \boldsymbol{q}} \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}\boldsymbol{g}} + \frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}} \frac{\mathrm{d}\dot{\boldsymbol{q}}}{\mathrm{d}\boldsymbol{g}} + \frac{\partial \mathbf{F}}{\partial \boldsymbol{g}} = 0 \text{ for } 0 \le t \le T, \text{ and } \frac{\partial \mathbf{G}}{\partial \boldsymbol{q}(0)} \frac{\mathrm{d}\boldsymbol{q}(0)}{\mathrm{d}\boldsymbol{g}} + \frac{\partial \mathbf{G}}{\partial \boldsymbol{g}} = 0. \tag{2.2}$$

Evaluating the gradient $\mathrm{d}\mathcal{J}/\mathrm{d}\boldsymbol{g}$ in equation (2.1) requires solving for the sensitivity $\mathrm{d}\boldsymbol{q}/\mathrm{d}\boldsymbol{g}$ using equation (2.2), which necessitates the resolution of a linear equation for each parameter g_i. It can be readily established that the computational cost of evaluating the gradient using the above equations is directly proportional to the number of design variables in $\boldsymbol{g}$, and, as the number of parameters increases, this direct approach reaches its limit. For this reason, an alternative strategy is required. An efficient procedure for the computation of sensitivities in a high-dimensional design space lies in the construction and use of adjoint equations. To this end, an auxiliary functional known as the augmented Lagrangian is defined as

$$\mathcal{L}(\boldsymbol{q}, \boldsymbol{g}, \boldsymbol{\lambda}, \boldsymbol{\lambda}_0) = \mathcal{J}(\boldsymbol{q}, \boldsymbol{g}) - \int_0^T \boldsymbol{\lambda}^\dagger \mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{g}, t)\, \mathrm{d}t - \boldsymbol{\lambda}_0^\dagger \mathbf{G}(\boldsymbol{q}(0), \boldsymbol{g}), \tag{3}$$

where the superscript $(\cdot)^\dagger$ denotes the conjugate transpose and $\boldsymbol{\lambda}(t)$ and $\boldsymbol{\lambda}_0$ are the so-called Lagrange multipliers or adjoint variables. Note that if the constraints in equation (1.2) hold, the value of the Lagrangian and the cost functional coincide, together with their respective gradients, for an arbitrary choice of $\boldsymbol{\lambda}(t)$ and $\boldsymbol{\lambda}_0$. We next look for specific choices of these variables such that the cost of computing the gradient $\mathrm{d}\mathcal{L}/\mathrm{d}\boldsymbol{g}(= \mathrm{d}\mathcal{J}/\mathrm{d}\boldsymbol{g})$ is independent of the number of design variables. We hence consider

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{g}} = \frac{\mathrm{d}\mathcal{J}}{\mathrm{d}\boldsymbol{g}} - \int_0^T \boldsymbol{\lambda}^\dagger \left(\frac{\partial \mathbf{F}}{\partial \boldsymbol{q}} \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}\boldsymbol{g}} + \frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}} \frac{\mathrm{d}\dot{\boldsymbol{q}}}{\mathrm{d}\boldsymbol{g}} + \frac{\partial \mathbf{F}}{\partial \boldsymbol{g}} \right) \mathrm{d}t - \boldsymbol{\lambda}_0^\dagger \left(\frac{\partial \mathbf{G}}{\partial \boldsymbol{q}(0)} \frac{\mathrm{d}\boldsymbol{q}(0)}{\mathrm{d}\boldsymbol{g}} + \frac{\partial \mathbf{G}}{\partial \boldsymbol{g}} \right) \tag{4}$$

and substitute equation (2.1) into equation (4). Integrating by parts the term $\boldsymbol{\lambda}^\dagger(\partial \mathbf{F}/\partial \dot{\boldsymbol{q}})(\mathrm{d}\boldsymbol{q}/\mathrm{d}\boldsymbol{g})$ we arrive, after some manipulations, at

$$\begin{aligned} \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{g}} = &\int_0^T \left[\left(\frac{\partial J}{\partial \boldsymbol{q}} - \boldsymbol{\lambda}^\dagger \frac{\partial \mathbf{F}}{\partial \boldsymbol{q}} + \frac{\partial}{\partial t} \left(\boldsymbol{\lambda}^\dagger \frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}} \right) \right) \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}\boldsymbol{g}} + \frac{\partial J}{\partial \boldsymbol{g}} - \boldsymbol{\lambda}^\dagger \frac{\partial \mathbf{F}}{\partial \boldsymbol{g}} \right] \mathrm{d}t - \\ &- \left[\boldsymbol{\lambda}^\dagger \frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}} \frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}\boldsymbol{g}} \right]_0^T - \boldsymbol{\lambda}_0^\dagger \left(\frac{\partial \mathbf{G}}{\partial \boldsymbol{q}(0)} \frac{\mathrm{d}\boldsymbol{q}(0)}{\mathrm{d}\boldsymbol{g}} + \frac{\partial \mathbf{G}}{\partial \boldsymbol{g}} \right). \end{aligned} \tag{5}$$

We notice that the gradient $\mathrm{d}\mathcal{L}/\mathrm{d}\boldsymbol{g}$ can be made independent of $\mathrm{d}\boldsymbol{q}/\mathrm{d}\boldsymbol{g}$ by setting

$$\frac{\partial J}{\partial \boldsymbol{q}} - \boldsymbol{\lambda}^\dagger \frac{\partial \mathbf{F}}{\partial \boldsymbol{q}} + \frac{\partial}{\partial t} \left(\boldsymbol{\lambda}^\dagger \frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}} \right) = 0 \text{ for } 0 \le t \le T, \tag{6.1}$$

$$\boldsymbol{\lambda}^\dagger(T) \left. \frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}} \right|_T = 0, \tag{6.2}$$

and

$$\boldsymbol{\lambda}^\dagger(0) \left. \frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}} \right|_0 - \boldsymbol{\lambda}_0^\dagger \frac{\partial \mathbf{G}}{\partial \boldsymbol{q}(0)} = 0. \tag{6.3}$$

We also note that the adjoint equation constitutes an algebraic-differential equation. The backward solution in effect provides gradient or sensitivity information for an optimization problem, using the optimality condition (referred to as the Karush-Kuhn-Tucker (KKT) equation) that links the result from the adjoint evolution equation to an optimal control strategy. Finally, the gradient reads

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}\boldsymbol{g}} = \int_0^T \left(\frac{\partial J}{\partial \boldsymbol{g}} - \boldsymbol{\lambda}^\dagger \frac{\partial \mathbf{F}}{\partial \boldsymbol{g}} \right) \mathrm{d}t - \boldsymbol{\lambda}_0^\dagger \frac{\partial \mathbf{G}}{\partial \boldsymbol{g}}. \tag{7}$$

Solving simultaneously for $\boldsymbol{g}$, $\boldsymbol{q}(t)$ and $\boldsymbol{\lambda}(t)$ using equations (1.2), (6), (7) and setting $\mathrm{d}\mathcal{J}/\mathrm{d}\boldsymbol{g} = 0$ leads to the so-called one-shot method. For the case of numerical simulations of unsteady flows, the resulting problem becomes computationally intractable due to the large dimensionality of the resulting system. For this reason, an iterative approach is usually preferred, where the evolution equations for the forward (equation (1.2)) and backward problems (equation (6)) are solved exactly, and the condition $\mathrm{d}\mathcal{J}/\mathrm{d}\boldsymbol{g} = 0$ is solved iteratively. This approach produces an increasingly optimal design or control effort, as the direct and adjoint equations are solved repeatedly. Since the adjoint equations are inherently linear, the application of parallel-in-time techniques is rather straightforward. Section 4 will therefore illustrate the application of one such parallel-in-time algorithm to the temporal evolution of the adjoint equation in order to reduce the computational effort for each optimization iteration.

3. Governing equations

In this section, the differential-algebraic equations that arise from the spatial discretization of the Navier–Stokes equations for incompressible flow are presented. Without loss of generality, we consider the projection-based immersed boundary method introduced by [36] as implemented in [37]. This method is proposed for incompressible flows over obstacles with prescribed surface motion. Following this approach, the physical domain is modified by embedding these obstacles and introducing localized volume forces at their boundaries such that the respective boundary conditions are satisfied. In continuous form, we have

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla p + \frac{1}{Re} \nabla^2 \boldsymbol{u} + \int_B \boldsymbol{f}(\boldsymbol{\xi}(\boldsymbol{s}, t)) \delta(\boldsymbol{\xi}(\boldsymbol{s}, t) - \boldsymbol{x}) \, \mathrm{d}\boldsymbol{s}, \tag{8.1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{8.2}$$

$$\boldsymbol{u}(\boldsymbol{\xi}(\boldsymbol{s}, t)) = \int_\Omega \boldsymbol{u}(\boldsymbol{x}, t) \delta(\boldsymbol{\xi}(\boldsymbol{s}, t) - \boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \boldsymbol{u}_B(\boldsymbol{s}, t), \tag{8.3}$$

on the domain $\Omega \times [0, T]$, together with the initial conditions at $t = 0$, given by $\boldsymbol{u}(\boldsymbol{x}, 0) = \boldsymbol{u}_0$, and suitable boundary conditions at $\partial\Omega$. In the above, $\boldsymbol{u}$ and p are the non-dimensionalized velocity vector and kinematic pressure, respectively, and Re denotes the Reynolds number. The last term in equation (8.1) represents the contribution of the localized forces $\boldsymbol{f}$ at the surface of the obstacles B described by $\boldsymbol{\xi}(\boldsymbol{s}, t)$ and δ denotes the Dirac delta function. Similarly to the role of the pressure p in fulfilling the incompressibility constraint, equation (8.2), the localized forces $\boldsymbol{f}$ are introduced such that the velocity field at the boundary of the surfaces coincides with the prescribed value $\boldsymbol{u}_B(\boldsymbol{\xi}, t)$ (equation (8.3)). In the following, we consider initial and boundary conditions that are parameterized by the vector of design variables $\boldsymbol{g}$ that we seek to optimize according to a given objective function.

Equations (8.1)–(8.3) are then discretized in space using a finite-volume method on a Cartesian grid with a staggered arrangement for the velocity and pressure variables. A set of Lagrangian points is introduced at the surface of the obstacles, and the boundary forces are then applied on these points to satisfy the no-slip constraint along the surface of the immersed bodies. In particular, we have

$$\mathbf{F}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{g}, t) \equiv \begin{pmatrix} \mathbf{M}\dot{\boldsymbol{u}} + \mathbf{N}(\boldsymbol{u}, \boldsymbol{g}) + \mathbf{Q}\boldsymbol{\phi} - \mathbf{L}\boldsymbol{u} + \mathbf{b}_1(\boldsymbol{g}) \\ \mathbf{Q}^\dagger \boldsymbol{u} + \mathbf{b}_2(\boldsymbol{g}) \end{pmatrix} = 0 \text{ for } 0 \le t \le T, \tag{9.1}$$

$$\boldsymbol{u}(0) - \boldsymbol{u}_0(\mathbf{g}) = 0, \tag{9.2}$$

where $\boldsymbol{u}$ and $\boldsymbol{\phi} = (p, \boldsymbol{f})^T$ are now the discrete representation of the velocity, pressure and boundary forces, respectively. In the above, $\mathbf{M}$ is the mass matrix, $\mathbf{N}(\boldsymbol{u}, \boldsymbol{g})$ is the discrete advection operator and $\mathbf{L}$ represents the discretized diffusion operator. The operator $\mathbf{Q}$ stands for the discretized gradient and interpolation operators that are respectively applied to the pressure and the localized forces. Finally, the boundary terms $\mathbf{b}_1$ and $\mathbf{b}_2$ arise from the spatial discretization of the diffusion operator and the constraints, i.e. incompressibility and no-slip boundary condition at the obstacles. Note that in the above equations the dependency on the design variables has been indicated explicitly. The above system of equations can be recast into the form given in equation (1.2) by introducing the state vector $\boldsymbol{q} = (\boldsymbol{u}, \boldsymbol{\phi})^T$. The temporal discretization of the governing equations (forward problem) is carried out using the implicit Crank-Nicolson method for the viscous terms and the explicit second-order Adams-Bashforth scheme for the convective terms. The reader is referred to [36] for further details.

3.1. Adjoint equations

Traditionally, adjoint equations are derived from the continuous equations by applying a variational principle to the unconstrained optimization problem and setting the first variations with respect to all involved dependent variables to zero. This results in governing equations for the direct (original) and for the adjoint variables, together with appropriate boundary conditions, initial conditions, and optimality expressions, which subsequently have to be discretized and implemented [38–40]. This process, in particular for complex governing equations and/or optimization objectives, is rather cumbersome and error-prone. Alternatively, the spatially discretized equations (e.g. resulting from the application of the method of lines) can be used and processed by automatic-differentiation (AD) software to produce an associated adjoint code [41]. This approach often leads to overly inflated, and thus very inefficient and ultimately impractical, codes. Recently, Fosas de Pando et al. [42] implemented and validated an approach that extracts linearized and adjoint information directly from a nonlinear simulation code. Following this approach, the nonlinear modules are linearized and trans-conjugated, such that by producing the adjoint of a directed acyclic graph (a simple reversal procedure), the adjoint solution is produced. In this manner, the adjoint information is simply extracted from the already existing nonlinear simulation code, avoiding significant additional programming effort, and exploiting the discretization schemes of the original solver, causing it to execute as efficiently (and parallel) as the original code; the adjoint code is simply embedded in the nonlinear simulation code. In addition, modifications to the code, such as the addition of reactive-flow simulation capabilities, are automatically reflected on the adjoint side by local differentiation of the added modules/subroutines, usually by means of a complex-step differentiation. This strategy has shown great promise in reducing trailing-edge noise and improving airfoil shape design in an aeroacoustic application [42] as well as extracting the mechanism governing the frequency response of an M-flame to the surrounding acoustic wave [15], and has been adopted here for extracting the discretized adjoint equations.

Using the discrete form of the equations presented in (9), the adjoint equations, following the formalism presented in (6), are

$$\mathbf{M}\frac{\mathrm{d}\tilde{\boldsymbol{u}}}{\mathrm{d}t} - \left(-\mathbf{L} + \frac{\partial \mathbf{N}}{\partial \boldsymbol{u}}^{\dagger}\right)\tilde{\boldsymbol{u}} - \mathbf{Q}\tilde{\boldsymbol{\phi}} + \frac{\partial J}{\partial \boldsymbol{u}}^{\dagger} = 0, \tag{10.1}$$

$$-\mathbf{Q}^{\dagger}\tilde{\boldsymbol{u}} + \frac{\partial J}{\partial \boldsymbol{\phi}}^{\dagger} = 0 \text{ for } 0 \le t \le T, \tag{10.2}$$

$$\text{and } \tilde{\boldsymbol{u}}(T) = 0. \tag{10.3}$$

Note that the adjoint variable is $\boldsymbol{\lambda}^{\dagger} = (\tilde{\boldsymbol{u}}^{\dagger}\ \tilde{\boldsymbol{\phi}}^{\dagger})$, where $\tilde{\boldsymbol{u}}$ and $\tilde{\boldsymbol{\phi}}$ are, respectively, the adjoint velocity field and adjoint pressure and localized forces. In the derivation, it has been taken into account that $\mathbf{M}$ is independent of time and the operators $\mathbf{M}$ and $\mathbf{L}$ are symmetric. The adjoint equations are again a system of differential-algebraic equations, and they are integrated backwards in time starting from $t = T$ once the forward (direct) solution has been computed. With the two equations solved, the cost-functional gradient is given by

$$\frac{\mathrm{d}\mathcal{J}}{\mathrm{d}\boldsymbol{g}} = \int_0^T \left(\frac{\partial J}{\partial \boldsymbol{g}} - \tilde{\boldsymbol{u}}^{\dagger}\left(\frac{\mathrm{d}\mathbf{b}_1}{\mathrm{d}\boldsymbol{g}} + \frac{\partial \mathbf{N}}{\partial \boldsymbol{g}}\right) - \boldsymbol{\phi}^{\dagger}\frac{\mathrm{d}\mathbf{b}_2}{\mathrm{d}\boldsymbol{g}}\right)\mathrm{d}t + \tilde{\boldsymbol{u}}^{\dagger}(0)\mathbf{M}\frac{\mathrm{d}\boldsymbol{u}_0}{\mathrm{d}\boldsymbol{g}}. \tag{11}$$

4. Parallel-in-time algorithm

A brief account of the development and implementation of various parallel-in-time algorithms can be found in [31]. Following the overlapping time-domain decomposition method developed by Gander and Güttel *Paraexp* [34], the time domain of a linear initial-value problem is decomposed into smaller segments of constant size ΔT, and the problem is separated into subproblems on overlapping time intervals. Fig. 1 represents schematically the integration of the initial-value problem using the *Paraexp* algorithm on three different time partitions, assigned to three different threads $p = 1, 2, 3$. Each subproblem is subsequently split into homogeneous and inhomogeneous components. The inhomogeneous problems (red) with zero initial conditions are simultaneously solved in the corresponding time partition $t \in [T_{p-1}, T_p]$. The inhomogeneous solution at the end of each time segment is then used as initial conditions for the homogeneous problems (blue), which are then integrated on $t \in [T_p, T]$, where T signifies the final time horizon. The final result is a superposition of the final solutions of each segment.

Integrating the homogeneous problem in time can be expensive and time consuming, therefore, fast time integrators are essential in order to speed up the process. In *Paraexp*, this speed up is obtained by using exponential time integrators to solve the homogeneous subproblems.

Paraexp shows great promise due to the following advantages: this method performs particularly well if the existing inhomogeneity is difficult to integrate, which is a common scenario in complex unsteady flows. In addition, it allows the use of any existing serial time integration method. Moreover, this direct method is non-iterative and requires a single communication between threads at the end of the algorithm. As a result, the achieved parallel efficiency is higher than

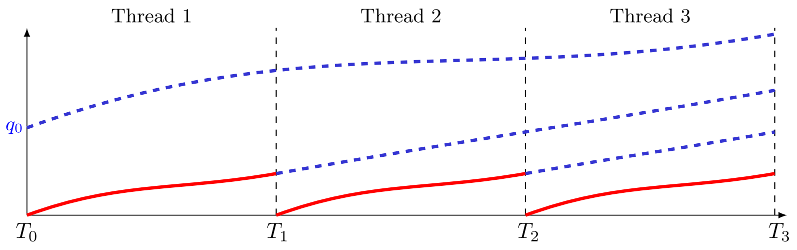


Fig. 1. Overlapping time decomposition of an initial-value problem into four inhomogeneous problems with zero initial guess (solid red curves) and four homogeneous problems (dashed blue curves), the latter are exponentially propagated. The solution of the original problem is obtained by summation of all these curves [34]. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

the maximal achievable parallel efficiency of the (Krylov-enhanced) parareal algorithms, and in particular the algorithm by Farhat et al. [43] for linear initial-value problems, which require more than a single iteration in general. Although *Paraexp* has been mainly applied to linear systems, an extension of the algorithm to a simplified nonlinear problem is now also available [44]. Due to these advantages, this method is employed in this study for time-parallelism of the adjoint equation.

Parallel-in-time algorithms have been mostly employed in accelerating the primal (forward) problem. Some applications to the optimization procedure are also available, in particular, in the recent work of Skene et al. [45], where algorithms are proposed based on the linear *Paraexp* algorithm developed by Güttel [34] and its extension to nonlinear partial differential equations (PDE)s by Kooij [46]. In their study Skene et al., investigated various strategies of accelerating both the forward (nonlinear) problem and the backward (linear) problem by using various combinations of *Paraexp* in its linear and nonlinear forms. Their findings, however, show that based on the form of the nonlinear forward problem the fully time-parallel algorithm can prove less effective due to the number of iterations required to converge the nonlinear version of the algorithm. Therefore, in this work, we concentrate on accelerating the linear part of the optimization algorithm concerned with time integration of the adjoint equations with algebraic constraint, resulting in a hybrid serial-direct-parallel-adjoint algorithm.

For this purpose, we consider the generic optimization problem (1), with equation (1.2) representing the direct problem and equation (6) denoting the corresponding adjoint problem. The direct problem is solved in serial on $p \in \{1, \dots, N\}$, while accounting for appropriate time partitioning $t \in [T_{p-1}, T_p]$, and the adjoint problem is split into its homogeneous and inhomogeneous components, as previously described in the *Paraexp* algorithm. The resulting inhomogeneous algebraic adjoint equations, in general form, are solved backward in time by thread $p \in \{N, \dots 1\}$ on $[T_p, T_{p-1}]$,

$$\frac{\partial J}{\partial \boldsymbol{q}} - \boldsymbol{\lambda}^{\dagger}_{I,p}\frac{\partial \mathbf{F}}{\partial \boldsymbol{q}} + \frac{\partial}{\partial t}\left(\boldsymbol{\lambda}^{\dagger}_{I,p}\frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}}\right) = 0 \tag{12.1}$$

$$\boldsymbol{\lambda}_{I,p}(T_p) = 0, \tag{12.2}$$

where $\boldsymbol{\lambda}_{I,p}$ is the inhomogeneous adjoint solution for thread p.

A schematic of the algorithm is given in Fig. 2, while a pseudo-code is presented in Algorithm 1. The direct problem is integrated forward in time on thread p up to T_p. The last time solution $\boldsymbol{q}(T_p)$ is then communicated to the next thread (Fig. 2a) is used as the initial condition to continue the forward integration of the direct problem on the corresponding time partition. While thread $p+1$ is solving the direct problem, the preceding threads solve the inhomogeneous adjoint problems (12.2), with zero initial condition, on $t \in [T_p, T_{p-1}]$ (Fig. 2b).

Following this scheme, when the direct problem is solved by the last thread $p = N$, the inhomogeneous equations are almost completely solved on the antecedent time partitions (Fig. 2c). A non-uniform time partitioning is used to ensure the simultaneity of the solutions of the direct and adjoint inhomogeneous problems, thus preventing latency between threads. The analytic expression for the time partition is given by

$$T_p = T\left(\frac{1-\left(\frac{k}{k+1}\right)^p}{1-\left(\frac{k}{k+1}\right)^N}\right), \tag{13}$$

where T_p is the final time of thread p, T denotes the total time horizon, N stands for the number of threads and $k = \tau_I^{\dagger}/\tau_I$ represents the ratio between the time taken per time unit for the inhomogeneous adjoint and the direct solver. Further details on the time partitioning can be found in [45].

Once the inhomogeneous equation is integrated down to T_{p-1} by thread p, this thread initializes and solves the homogeneous problem, given below, for the time partition $[T_{p-1}, 0]$,

$$-\boldsymbol{\lambda}^{\dagger}_{H,p}\frac{\partial \mathbf{F}}{\partial \boldsymbol{q}} + \frac{\partial}{\partial t}\left(\boldsymbol{\lambda}^{\dagger}_{H,p}\frac{\partial \mathbf{F}}{\partial \dot{\boldsymbol{q}}}\right) = 0, \tag{14.1}$$

$$\boldsymbol{\lambda}_{H,p}(T_{p-1}) = \boldsymbol{\lambda}_{I,p}(T_{p-1}), \tag{14.2}$$

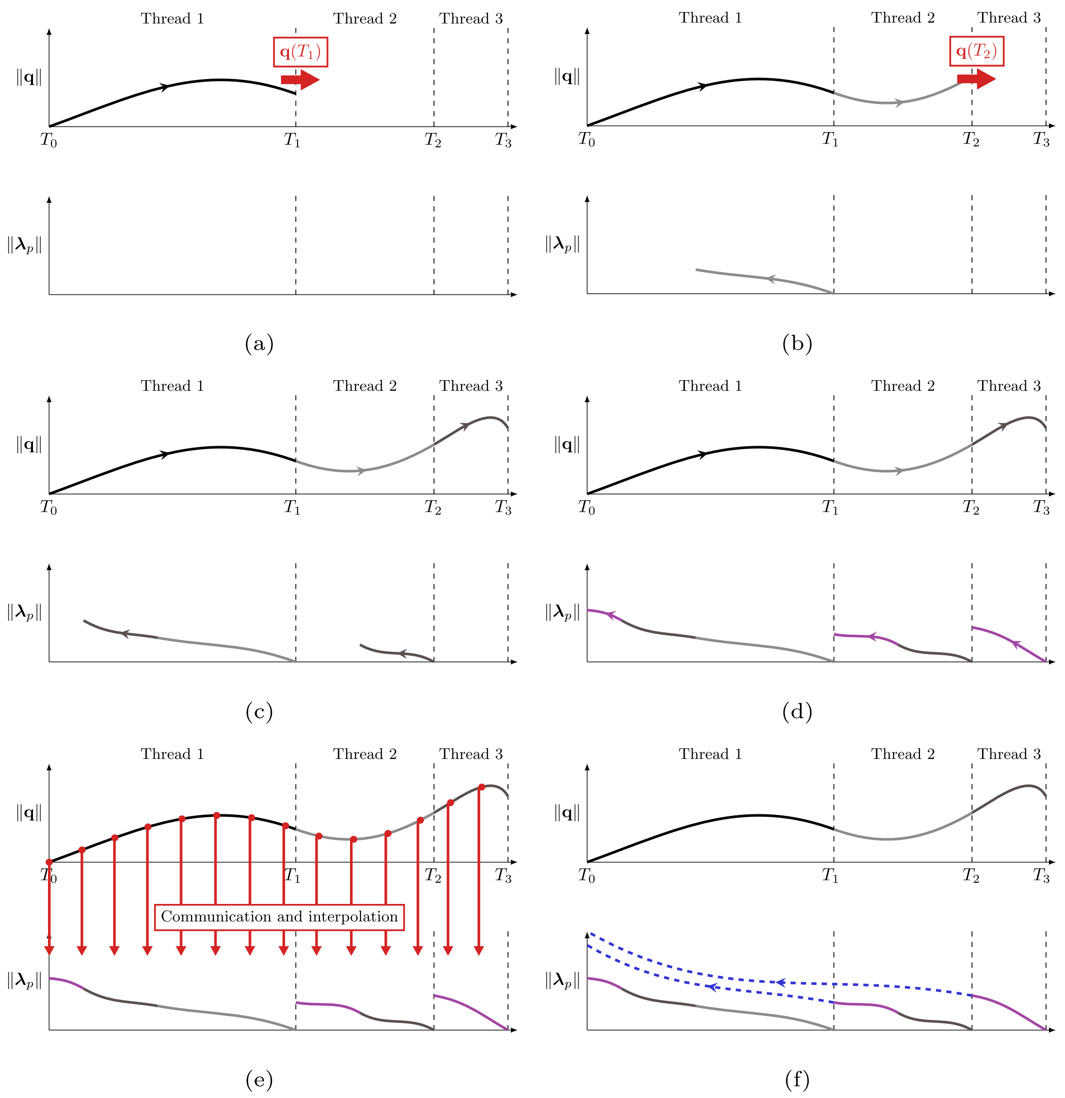


Fig. 2. A schematic of a direct-adjoint loop with three threads, using a parallel-in-time procedure. Solid lines indicate the forward evolution of the direct equation (top) and backward integration of the inhomogeneous adjoint equations on the three threads (bottom). Once the direct and inhomogeneous adjoint equations are solved, the direct solution is communicated (see red lines) to each thread to initialize the homogeneous adjoint equation. This equation is then solved up to T_0 (see dashed blue lines) by each thread.

with $\boldsymbol{\lambda}_{H,p}$ the homogeneous adjoint solution for thread p, and $\boldsymbol{\lambda}_{I,p}(T_{p-1})$ the solution at the final time of the inhomogeneous adjoint problem, here used as the initial condition.

The resolution of the adjoint problem requires the knowledge of the direct state at each time step. While the inhomogeneous equation is solved on the same time partition as the forward problem and maintains the same discretization scheme, the homogeneous equation is solved on $\left[0, T_{p-1}\right]$. Due to the time partitioning previously described, the generic thread p does not have access to the full direct solution up until $T = 0$. This solution must therefore be distributed to all threads, resulting in an unavoidable increase of time to solution. To reduce this one-time cost, each thread communicates a fraction of the direct solution to all. The submatrices are then stacked together and used by each thread to perform a linear interpolation on the coarser exponential time grid (Fig. 2e). The $N-1$ homogeneous adjoint equations are solved simultaneously (Fig. 2f).

An alternative way to circumvent memory cost, associated with the storage and communication of the forward solution, involves checkpointing algorithms, where only a small number of time steps are stored. Following this approach, the direct equation is integrated in time, saving the solution only at the checkpoints chosen by the underlying algorithm [47]. The direct-adjoint loops are then solved using the parallel-in-time algorithm between checkpoints, saving all intermediate solutions. The values of the cost functional and gradient can be computed between checkpoints, further reducing memory cost. The incorporation of checkpointing schemes on the proposed parallel-in-time algorithm is straightforward, and its perfor-

mance using equispaced checkpoints has been studied in detail in [45]; it therefore will not be addressed here.
The total time to solution, excluding the use of checkpointing, is thus

$$T^N = T\tau_I + (T_N - T_{N-1})\,\tau_I^\dagger + T_{N-1}\tau_H^\dagger \quad , \tag{15}$$

where N is the number of threads, τ_I the time to solution needed per time unit to solve the inhomogeneous direct equation, and $\tau_I^\dagger$ and $\tau_H^\dagger$ are the time to solution needed per time unit to solve the inhomogeneous and homogeneous adjoint equations, respectively. The efficiency of this parallel-in-time algorithm crucially relies on the minimization of the last two terms in (15), representing the time needed to solve the inhomogeneous adjoint problem on the last time partition and the time to solve the homogeneous adjoint problems. As observed in Fig. 2d, the non-uniform time partitioning assigns a smaller time interval to the last thread, so that the time required to solve the inhomogeneous adjoint on this time partition is ensured to be negligible. Finally, the cost associated with the resolution of the homogeneous adjoint problems can be reduced using an exponential time integrator, as implemented in *Paraexp*. Exponential time integrators and their extensions to algebraic differential equations will be discussed in detail in section 4.1.

The original *Paraexp* computes the total solution as a superposition of the solution computed by each thread, however, in cases where the size of the problem is large, the resulting communication can become rather expensive. When applied to the adjoint equation, the knowledge of the total solution is required only to compute the gradient. As a result, the partial gradients can be computed locally by each core using the optimality condition (11) and finally distributed to all threads and summed. This approach is employed both for the gradient and the value of the cost functional and reduces the size of the communication considerably.

The application of this parallel-in-time strategy to the problem of interest to this work, presented by equation (10.3), results in the following semidiscrete inhomogeneous and homogeneous set of equations (for a generic thread p) respectively,

$$\mathbf{M}^\dagger\dot{\tilde{\boldsymbol{u}}}_{I,p} + \left(\mathbf{N}_u(\boldsymbol{u})^\dagger - \mathbf{L} + \mathbf{b}_{1,u}^\dagger(\boldsymbol{u},\mathbf{g})\right)\tilde{\boldsymbol{u}}_{I,p} + \mathbf{Q}\tilde{\boldsymbol{\phi}}_{I,p} + \mathbf{J}_u^\dagger = 0 \tag{16.1}$$

$$\mathbf{Q}^\dagger\tilde{\boldsymbol{u}}_{I,p} - J_\phi^\dagger\tilde{\boldsymbol{\phi}}_{I,p} = 0 \text{ for } T_{p-1} \le t \le T_p, \quad \text{and} \quad \boldsymbol{\lambda}_{\boldsymbol{I}}(T_p) = 0, \tag{16.2}$$

and

$$\mathbf{M}^\dagger\dot{\tilde{\boldsymbol{u}}}_{H,p} + \left(\mathbf{N}_u(\boldsymbol{u})^\dagger - \mathbf{L}\right)\tilde{\boldsymbol{u}}_{H,p} + \mathbf{Q}\tilde{\boldsymbol{\phi}}_{H,p} = 0 \tag{16.3}$$

$$\mathbf{Q}^\dagger\tilde{\boldsymbol{u}}_{H,p} = 0 \text{ for } 0 \le t \le T_{p-1}, \quad \text{and} \quad \boldsymbol{\lambda}_{H,p}(T_{p-1}) = \boldsymbol{\lambda}_{I,p}(T_{p-1}), \tag{16.4}$$

where discrete operators have been introduced in equation (10.3), the subscripts I and H denote the inhomogeneous and homogeneous adjoint variables, respectively, and subscript p indicates the thread.

Algorithm 1 Parallel-in-time direct-adjoint loop.

```
    Input: $\boldsymbol{q}(0)$, $\mathbf{g}$, $T$, $N$
    Output: $\mathcal{J}(\mathbf{q},\mathbf{g})$, $\mathrm{d}\mathcal{J}/\mathrm{d}\mathbf{g}$
 1: Compute $k = \tau_I^\dagger/\tau_I$
 2: Define $N$ non uniform time partitions for $t \in [0,T]$ using equation (13)
 3: for $p = 0,\dots,N-1$ do
 4:     if $p > 0$ then
 5:         Wait to receive $\boldsymbol{q}_{p-1}(T_p)$ from thread $p-1$          ▷ communication
 6:     end if
 7:     Solve direct problem (1.2) on $[T_p, T_{p+1}]$
 8:     if $p < N-1$ then
 9:         Send $\boldsymbol{q}_p(T_{p+1})$ to thread $p+1$                     ▷ communication
10:     end if
11:     Solve inhomogeneous adjoint problem (12) on $[T_{p+1}, T_p]$
12: end for
13: Scatter a fraction of the direct solution to all                          ▷ communication
14: for $p = 1,\dots,N-1$ do
15:     Interpolate direct solution on $[0, T_p]$
16:     Solve homogeneous adjoint problem (14) on $[0, T_p]$
17:     Stack $\boldsymbol{\lambda}_{H,p}$ and $\boldsymbol{\lambda}_{I,p}$ to form full adjoint solution
18:     Compute cost functional on each thread $p$, $\mathcal{J}_p(\mathbf{q}_p,\mathbf{g})$
19:     Compute gradient on each thread $p$, $\mathrm{d}\mathcal{J}_p/\mathrm{d}\mathbf{g}$
20: end for
21: Sum cost $\mathcal{J}(\mathbf{q},\mathbf{g}) = \sum_p \mathcal{J}_p$                       ▷ communication
22: Sum gradient $\mathrm{d}\mathcal{J}/\mathrm{d}\mathbf{g} = \sum_p \mathrm{d}\mathcal{J}_p/\mathrm{d}\mathbf{g}$              ▷ communication
```

4.1. Exponential time integrators

As mentioned in the previous section, the bottleneck of time acceleration using the proposed parallel-in-time procedure is the cost associated with integrating the homogeneous problem on each thread (Algorithm 1 line 16). Exponential time integrators can be employed to reduce this cost due to their superior accuracy with a minimal number of time-steps, and are directly applicable since the solution of the homogeneous problems can be expressed analytically in terms of the matrix exponential of the state-matrix.

Exponential integrators rely on numerical approximations of the matrix exponential in order to perform time-stepping. Considering a general form of a linear initial value problem, and the corresponding initial condition

$$\begin{aligned}\frac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} &= \mathbf{A}\boldsymbol{q} \\ \boldsymbol{q}(0) &= \boldsymbol{q}_0,\end{aligned} \tag{17}$$

the solution of this equation can be expressed in terms of the matrix exponential of the state matrix as

$$\boldsymbol{q} = \exp\left[(\Delta t)\,\mathbf{A}\right]\boldsymbol{q}_0 \quad , \tag{18}$$

where $\mathbf{A}$ is the system state matrix (or Jacobian). Here, matrix $\mathbf{A}$ is assumed to be time independent. Direct access to $\mathbf{A}$ becomes challenging as the dimension of the discretized problem increases. In such cases the exponential matrix is approximated directly using two categories of methods: projection-based methods and polynomial-interpolation-based methods. Projection-based methods approximate the matrix exponential on the orthogonal, lower-dimensional Krylov subspace of $\mathbf{A}$ [48], using the Arnoldi Algorithm 2 [49,50].

The homogeneous algebraic adjoint equation presented in (14) has to be reformulated for an exponential time integrator to be applicable. An additional projection procedure needs to be devised to include the constraint in equation (14.1) and to reformulate the equation in the form of a partial differential equation presented in (17) (without a constraint).

The divergence-free constraint of the incompressible formulation of the governing equations, along with the constraint formed by no-slip conditions on the immersed boundaries makes the application of such schemes non-trivial. To accommodate these constraints, the momentum equation must be reformulated [51]. To this end, the semi-discrete homogeneous adjoint system given in (16.3) and (16.4), for a generic process p, with appropriate initial conditions, multiplied by $\mathbf{Q}^\dagger\mathbf{M}^{\dagger-1}$ and differentiated with respect to time, gives

$$\mathbf{Q}^\dagger\dot{\tilde{\boldsymbol{u}}}_{H,n} = \mathbf{Q}^\dagger\mathbf{M}^{\dagger-1}\left(\mathbf{L}-\mathbf{N}_u(\boldsymbol{u}_n)^\dagger\right)\tilde{\boldsymbol{u}}_{H,n} + \mathbf{Q}^\dagger\mathbf{M}^{\dagger-1}\mathbf{Q}\tilde{\boldsymbol{\phi}}_{H,n} \tag{19.1}$$

$$\mathbf{Q}^\dagger\tilde{\boldsymbol{u}}_{H,n} = 0 \ . \tag{19.2}$$

Substituting (19.2) in (19.1) and solving for $\tilde{\boldsymbol{\phi}}_{H,n}$, results in

$$\tilde{\boldsymbol{\phi}}_{H,n} = -\left(\mathbf{Q}^\dagger\mathbf{M}^{\dagger-1}\mathbf{Q}\right)^{-1}\mathbf{Q}^\dagger\mathbf{M}^{\dagger-1}\left(\mathbf{L}-\mathbf{N}_u(\boldsymbol{u}_n)^\dagger\right)\tilde{\boldsymbol{u}}_{H,n} \ . \tag{20}$$

This formulation for $\tilde{\boldsymbol{\phi}}_{H,n}$ is then used in (19.1) to get the projection of the momentum equation into the subspace defined by the constraints (19.2). The homogeneous adjoint equation (16.3) and its projected initial condition are then rewritten as

$$\begin{aligned}\dot{\tilde{\boldsymbol{u}}}_{H,n} &= \mathbf{P}\left(\mathbf{L}-\mathbf{N}_u(\boldsymbol{u}_n)^\dagger\right)\tilde{\boldsymbol{u}}_{H,n} \\ \tilde{\boldsymbol{u}}_{H,n}(T_p) &= \mathbf{P}\,\tilde{\boldsymbol{u}}_{In}(T_p)\end{aligned} \tag{21}$$

with the associated projection operator

$$\mathbf{P} = \left[\mathbf{M}^{\dagger-1} - \mathbf{M}^{\dagger-1}\mathbf{Q}^\dagger\left(\mathbf{Q}^\dagger\mathbf{M}^{\dagger-1}\mathbf{Q}\right)^{-1}\mathbf{Q}^\dagger\mathbf{M}^{\dagger-1}\right]. \tag{22}$$

Using this projection (21) allows the application of an exponential integrator. It should however be noted that the right-hand side of equation (21) depends on time, whereas the system matrix $\mathbf{A} = \mathbf{P}\left(\mathbf{L}-\mathbf{N}_u(\boldsymbol{u}_n)^\dagger\right)$ is considered to be independent of time. In order to use the exact solution given in (18), the transconjugate advection operator $\mathbf{N}_u(\boldsymbol{u}_n)^\dagger$ is updated at each time iteration and is considered piecewise-constant in time. The accuracy of the method is further discussed in section 5. To avoid excessive memory costs, the approximation of the matrix exponential is accomplished by Krylov projection, which allows a matrix-free implementation. Additionally, the computation and storage of the projection operator in (22) is not practical. Therefore, a fractional step method [52] has to be added to the Arnoldi algorithm every time the product $\mathbf{A}\mathbf{v}_j$ needs to be computed (see Algorithm 2).

Algorithm 2 Arnoldi algorithm.

```
1: Given $\mathbf{v}_1$, with $\|\mathbf{v}_1\| = 1$
2: for $j = 1, \ldots, m$ do
3:     for $i = 1, \ldots, m$ do
4:         $h_{ij} = (\mathbf{A}\mathbf{v}_j, \mathbf{v}_i)$
5:     end for
6:     $\mathbf{w}_j = \mathbf{A}\mathbf{v}_j - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i$
7:     $h_{j+1,j} = \|\mathbf{w}_j\|$
8:     if $h_{j+1,j} = 0$ then
9:         stop
10:    end if
11:    $\mathbf{v}_{j+1} = \mathbf{w}_j / h_{j+1,j}$
12: end for
```

5. Results

In this section, the performance of the parallel-in-time adjoint algorithm is presented using a selection of cases, from drag reduction of a flow around a cylinder to reducing pressure loss across a blade using boundary control. In the first case, steady actuation is imposed using immersed boundary forces and in the second case, unsteady actuation is performed at a domain boundary, introducing new challenges for the parallel-in-time algorithm, which will be discussed in the following. The efficiency of the parallelization and the decrease of the computational cost has been evaluated for a single gradient extraction (one direct-adjoint loop).

The parallel-in-time algorithm has been implemented in the incompressible Navier-Stokes solver [37] using the *MPI for Python* package [53]. The subroutine has been structured to return only the values of the cost function and gradient and remains independent of the choice of a descent algorithm. In what follows, we used the L-BFGS-B method as implemented in the *scipy.optimize.minimize* class from SciPy [54]. In order to provide a fair comparison of the time-to-solution for the serial and parallel mode, the convergence of the descent algorithm has been fixed by equalizing the maximum number of iterations in both cases.

5.1. Temporal energy growth – examining the exponential time integrator

Before discussing the performance of the parallel-in-time optimization algorithm, the convergence of the Krylov-based exponential time integrator is assessed and compared to the original explicit second-order Adams-Bashforth method. As mentioned in the previous sections, the exponential time integrator is applied to the homogeneous part of the equation only. Therefore, the optimization problem has to be designed such that the resulting adjoint equation becomes homogeneous and can be integrated by either integration method independently. Optimizing the initial condition of the forward flow solver, such that an energy norm, $G(T)$, at a selected time T is maximized offers an ideal test case. In this approach, the optimization procedure aims at maximizing the ratio between an energy norm at $t = T$ and $t = 0$, resulting in the following cost functional

$$\mathcal{J}(\mathbf{q}, \mathbf{g}) = \frac{(\mathbf{g} \cdot \mathbf{g})}{(\mathbf{q}(T) \cdot \mathbf{q}(T))} \tag{23}$$

where $\mathbf{g} = \mathbf{q}(0)$ is the control parameter (the optimal initial condition), and the energy is computed with a simple L_2-norm, using the full state vector. Since the cost functional does not depend on the time evolution of the forward problem, the first term of the adjoint equation (6) is equal to zero, resulting in a homogeneous problem. The exponential integrator can therefore be used to propagate the entire equation backward in time.

The optimization is applied to a case of a lid driven cavity at $Re = 1000$ on a very short time interval in order to analyze the convergence of the two time integrators. It should be noted, however, that due to the presence of the convection term (a nonlinear operator), the solution of the adjoint problem is dependent on the forward problem. In order to remove the errors due to the forward integration of the primal problem from the convergence of the adjoint equation, the primal problem is integrated once using the most refined Δt. This solution is then stored and used to linearize the advection operator for the adjoint equations when the time step is increased. The convergence results obtained using the exponential integrator and the explicit Adam-Bashforth method are displayed in Fig. 3. The L_2-error is computed using the difference between the final solution of the adjoint equation $q^{\dagger}(0)$, from each time integration, and the reference solution, from a highly refined simulation. The optimization is performed without time parallelism, in order to focus on the time integrator. The errors due to the parallel-in-time algorithm will be assessed in the following sections.

This figure shows the explicit Adam-Bashforth method converging with a first-order slope, which is less than the expected second-order convergence. This deterioration is caused by the presence of immersed boundary forces and the use of fractional step to solve the direct problem. As expected, the exponential integrator shows a convergence rate that increases when Δt is refined. This behavior allows the use of a coarser time-step without affecting the accuracy of the final solution.

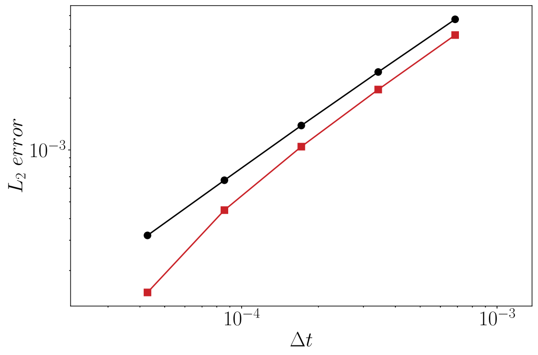


Fig. 3. Convergence history of time integrators: (black) Explicit Adam-Bashforth method; (red) Krylov-based exponential Euler method.

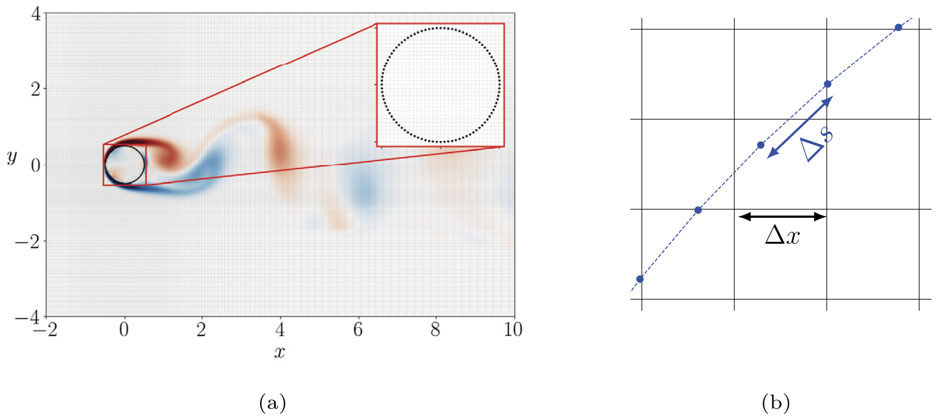


Fig. 4. Flow around a cylinder at $Re = 200$. (a) Vorticity contours and immersed boundary points. (b) Schematic of Lagrangian points (blue) versus the background Cartesian grid.

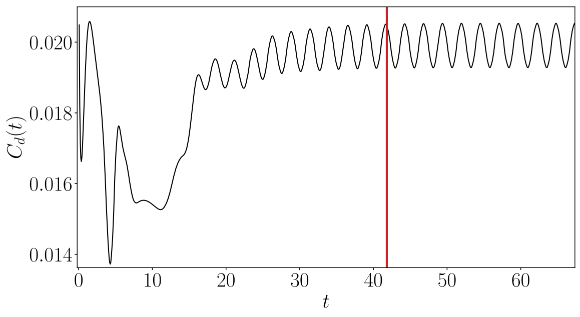


Fig. 5. Drag coefficient C_d, with the red line signifying the location where actuation starts, $t = 41$ s.

5.2. Drag reduction – steady actuation

The first system considered here is the two-dimensional flow around a cylinder at $Re = 200$. This regime is characterized by the appearance of a von Karman vortex street, and the vorticity contours of the uncontrolled case are shown in Fig. 4a, along with the immersed boundary points defining the surface of the cylinder. The spatial discretization of the domain must be uniform and refined in the proximity of the cylinder to ensure the stability of the solution [55], and the arc length Δs between the Lagrange points is selected to be equal to the size of the neighboring cells Δx, as shown in Fig. 4b. The system is discretized over a Cartesian non-uniform staggered grid of size $n_x = 382 \times 382$ and the surface of the cylinder is defined by $n_{IB} = 96$ Lagrangian points. The dimension of the state vector, including the immersed boundary points, is $n = 437581$, where $n = n_u + n_v + n_p + 2n_{IB}$ represents, respectively, the dimension of the discretized horizontal and vertical velocity, along with the pressure on the Cartesian grid, as well as horizontal and vertical velocities on the surface points.

The evolution of the drag coefficient is shown in Fig. 5. As expected in this regime, after the initial transient, the drag coefficient converges to the average value of 0.0187 with oscillations of amplitude 0.0005 and a frequency corresponding

Table 1
Value of the cost functional before and after the actuation process.

	$\mathbf{g} = 0$	$\mathbf{g} = \mathbf{g}_{opt}$
$\mathcal{J}(\mathbf{q}, \mathbf{g})$	3.987	1.700

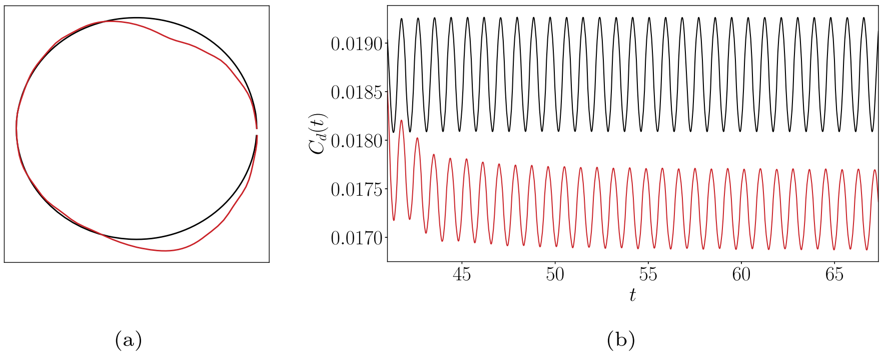


Fig. 6. The resulting actuation and cost functional. (a) Profile of the boundary velocities (red) projected on the cylinder surface as a reference (black). (b) Drag coefficient C_d without the actuation (black) and with the optimal control (red).

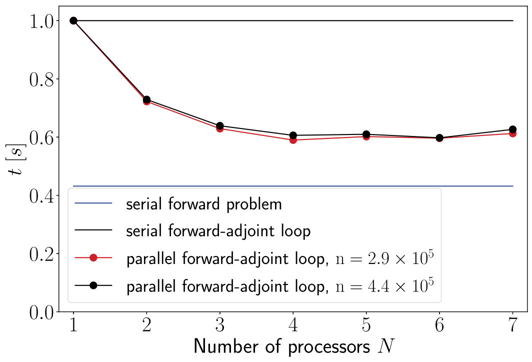


Fig. 7. Performance of the parallel-in-time algorithm (red line) compared to the serial counterpart (black line) reported for a single iteration of the optimization loop, using steady control.

to the shedding frequency of the cylinder $St = 0.196$. Once the simulation reaches steady state, $t \approx 41$, the optimization procedure is initiated and carried out over five shedding periods – the longest time horizon achievable without resorting to checkpointing algorithms.

A blowing/suction control is induced on the surface of the cylinder, allowing the vertical and horizontal velocities at each Lagrangian point to act as actuators. A smoothing filter is applied on the resulting velocity profiles to avoid discontinuities between neighboring points. The discrete cost functional is then defined as the sum of the squares of the drag coefficient $C_d = 2 f_x$, where f_x are the dimensionless boundary forces in the streamwise direction. In addition, a penalization of the control variables $\mathbf{g}$ is added to keep their value sufficiently small, resulting in

$$\mathcal{J}(\mathbf{q}, \mathbf{g}) = \frac{1}{T} \int_0^T C_d^2 \, dt + \alpha \, \|\mathbf{g}\|^2. \tag{24}$$

The solution of the optimization process is presented in Fig. 6. The resulting velocity profile (shown along the surface of the cylinder for clarity) in Fig. 6a shows blowing and suction effects on the back of the cylinder. The actuation results in a reduction of the objective functional $J(\mathbf{q}, \mathbf{g})$ and of the amplitude of the drag coefficient oscillations in time $C_d(t)$, respectively presented in Table 1 and Fig. 6b. The actuation reduces not only the average drag but also the amplitude of the oscillations around this average. There is a slight change in the frequency of the oscillations due to the steady actuation.

The performance of the parallel-in-time algorithm is shown in Fig. 7. Here, the total time to solution is represented for both the serial and the parallel cases.

For a direct comparison, no checkpointing algorithm is included (the solution of the forward integration at each time-step is accessible on memory). As a result, the time necessary to integrate the adjoint equation “in serial” is the same as that

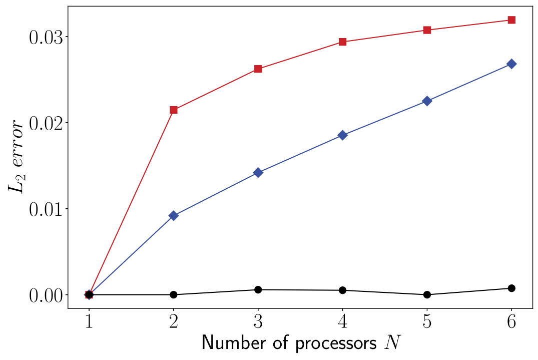


Fig. 8. Accuracy of the estimated gradient computed for a single optimization loop: —, Adams-Bashforth method; —, exponential integrator without interpolation; — Exponential integrator with interpolation.

of the forward problem, leading to twice the cost. To assess the performance of the algorithm with respect to problem size, we consider two spatial grids of dimension $n_x = 382 \times 382$ and $n_x = 252 \times 382$, resulting in two state vectors of dimensions $n = 437581$ and $n = 289877$, respectively. To compare the results obtained for the two cases, the figure has been normalized according to the total time needed to solve the direct-adjoint loop in serial.

The figure shows that, for both cases, using the parallel-in-time algorithm, the time needed to compute the gradient converges to a value very close to the time needed for simply solving the direct problem, as the number of threads is increased. In other words, the value of the cost functional and the gradient are obtained almost at the same time as the end of the forward integration, resulting in a reduction of 70% of the total time of the optimization process. Though the algorithm has been shown to be independent of the spatial discretization, its efficiency nonetheless depends on the selected time horizon for the optimization problem. As mentioned in section 4, the time partitioning is non-uniform and each thread solves the direct problem and the inhomogeneous adjoint on a shorter time interval than the previous thread. Increasing the number of threads involved in the resolution of the system reduces the number of time steps assigned to the last threads, reaching a limit where the time partition is smaller than the minimum time-step needed to solve the equation. In the case of the cylinder, the maximum number of threads that can be used is $N = 7$. When eight threads are used, the algorithm assigns three time-steps to the last core, which is the minimum allowed to form the advection term in equation (9). For $N > 4$ the figure shows a loss of efficiency of the algorithm, beyond this point, the time needed to communicate the direct solution to each core overwhelms the time saved by time parallelism.

The accuracy of the resulting gradient using the parallel-in-time algorithm is shown in Fig. 8. The gradient obtained using the serial adjoint algorithm (without the parallel-in-time treatment) is used as the reference in computing the error. Two major differences of the parallel-in-time implementation and its serial counterpart are that, firstly, in the parallel-in-time algorithm, the problem is separated into the homogeneous and inhomogeneous equations and an exponential integrator is used to propagate the solution of the homogeneous problem backward in time. In addition, in order to speed-up the adjoint loop, larger time steps are used to integrate the homogeneous problem. The latter step necessitates an interpolation of the forward problem from a finer resolution in time (used by the lower-order time integrator) on the coarser grid used by the exponential time integrator. This interpolation can impact the accuracy of the final optimization algorithm. In order to assess the ramification of each of these steps on the overall accuracy, three different implementations of the parallel-in-time algorithm are compared in Fig. 8.

As described in section 4, the parallel-in-time algorithm includes the communication of only a fraction of the direct solution to all tasks (see Fig. 2e), followed by an interpolation on each time partition. In order to remove the error associated with this interpolation, and to evaluate only the error due to the different time-integrators used to advance the adjoint equations, the assessment of the gradient accuracy has been performed over three shedding periods, allowing communication of the full direct solution without incurring memory errors.

In the first implementation, highlighted by the black line in the figure, the homogeneous adjoint equations are solved using the same time integrator and time discretization as the inhomogeneous equations. Therefore, the error accrued due to the interpolation step and the change in the time integrator is eliminated. As expected, the final solution remains the same as the original gradient, independent of the number of threads. In the second implementation, denoted by the blue line, the homogeneous equations are solved using the exponential time integrator but the same time discretization as the inhomogeneous problem, removing the error due to interpolation. The exponential integrator is more accurate than its counterpart resulting in a small error between the two solutions. This error increases as the number of threads increase since a larger portion of the problem is solved using the exponential integrator. Finally, in the last implementation (the implementation suggested in this study), denoted by the red line, the homogeneous equations are solved using the exponential time integrator on larger time intervals compared to the inhomogeneous equations. This figure shows that the error due to interpolation (the difference between the blue and red curves) decreases when increasing the number of threads. Due to a non-uniform

Table 2
Values of the cost functional and time to solution obtained running the same optimization problem in serial $N = 1$ and in parallel $N = 4$. The number of iterations to convergence of the optimization algorithm is fixed to three.

	Serial $N = 1$	Parallel $N = 4$
$\mathcal{J}(\mathbf{q}, \mathbf{g})$	1.666	1.669
$\lVert \mathrm{d}\mathcal{J}/\mathrm{d}\mathbf{g} \rVert$	2.87×10^{-5}	2.69×10^{-5}
t [s]	2118	1346

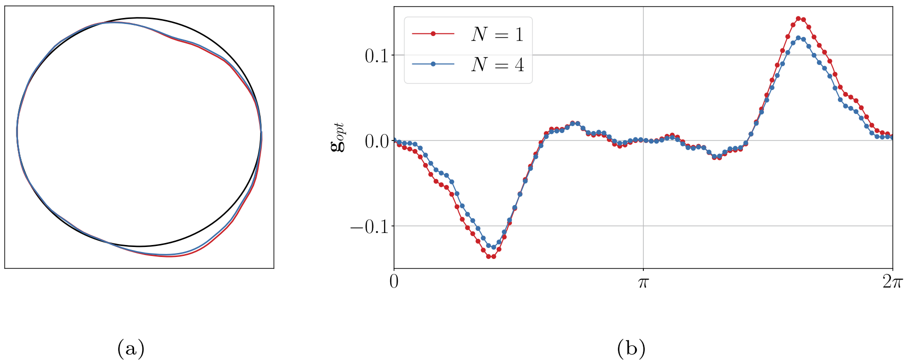


Fig. 9. (a) Profile of the boundary velocities obtained running the optimization in serial (red) and parallel (blue) projected on the cylinder surface as a reference (black). (b) Boundary velocities obtained running the optimization in serial (red) and parallel (blue).

time partitioning, smaller portions of the time domain are integrated using the exponential time integrator as the number of threads increase, causing the error due to interpolation to saturate.

To evaluate the effect of the error associated with exponential time integration on the optimal solution, the full optimization problem is solved both in serial and in parallel with $N = 4$. The serial and the parallel implementations are initialized with the same initial value of $\mathbf{g}$. When run in serial, the optimization problem reaches the optimal solution in three gradient-descent iterations. In order to compare fairly the time-to-solution and the results obtained, the number of iterations of the L-BFGS-B method is fixed to three, identical to the parallel simulation. Table 2 shows that the two different simulations converge approximately to the same minimum of the cost functional (24), reaching a small gradient value in both cases. However, the time to solution needed to solve the problem in parallel is only 63% of the time needed to solve the same problem in serial. The resulting actuation on the boundary velocities for the two cases is shown in Fig. 9, showing similar profiles. In particular, Fig. 9a shows the boundary velocities projected onto the cylinder, indicating blowing and suction effects on the surface, while Fig. 9b reports the actual values of the velocity at each Lagrangian point.

5.3. *Total pressure loss – unsteady actuation*

One of the main causes of aerodynamic losses in turbomachinary is due to vortices generated at the tip of the blades as they interact with the outside casing [56]. These vortices are promoted by the pressure gradient on the surface of the blades, as well as the relative motion between the blade tip and the casing of the rotor. In this section, we present an optimization problem with unsteady actuation inspired by this phenomenon. The objective is to determine whether modifications of the casing of the rotor (in the form of a small perturbation or roughness) would be able to suppress the generation of such vortices. This actuation is of relevance since it is reproducible in real-scale applications with relatively minimal effort.

Fig. 10 shows the rotor of an axial compressor that rotates with a certain angular velocity around the axis. In order to simplify the description of this complex phenomenon, the domain considered for the numerical simulations consists of a section of a single unit of the casing and blade geometry, defined as a square domain with periodic boundary conditions in the streamwise direction, as shown in the sketch of Fig. 10. The blade is represented by a vertical wall defined by the immersed boundaries and covers 90% of the vertical direction. Periodic boundary conditions in the streamwise direction simulate the series of blades. Curvature effects are neglected in this setup. The relative motion between the casing and the blade is enforced by applying a uniform horizontal velocity on the top boundary (casing).

Perturbations on the casing are replicated by adding a roughness on the top boundary. In order to reduce the dimension of the optimization problem, the roughness is considered to have the shape of a Gaussian function, and its width and amplitude is optimized via the optimization process. In addition, the roughness is assumed to have the characteristics of a dynamic roughness element.

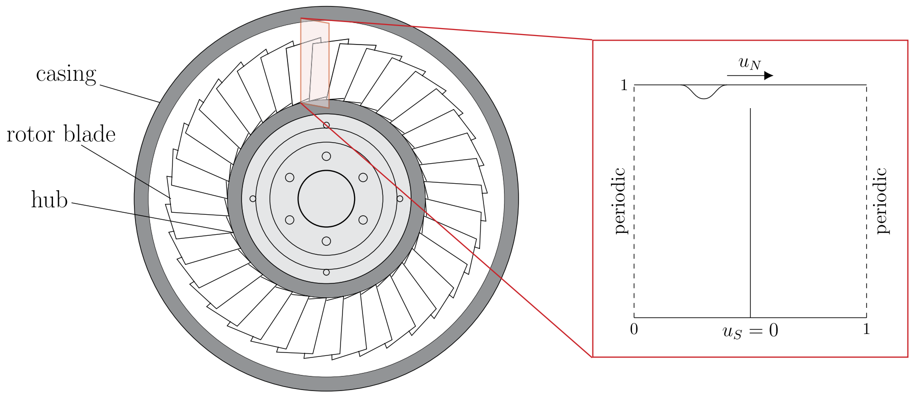


Fig. 10. Schematic of a rotor of an axial compressor, and the numerical domain in the red box with one blade (periodic boundary conditions on the horizontal axis simulate a series of blades). The upper boundary moves with a constant velocity and the dynamic roughness is added on its profile.

Table 3
Pressure loss minimization, listing the cost functional before and after the actuation.

	$\mathbf{g} = [0, 0]$	$\mathbf{g} = [0.095, 0.184]$
$\mathcal{J}(\mathbf{q}, \mathbf{g})$	2.000	1.982

Dynamic roughness elements have been investigated by [57] and further explored by [58] and [59]. In this approach, the roughness elements are modeled using linearized boundary conditions representing oscillating bumps with simple geometries. The roughness element is approximated by the streamwise and wall-normal velocity distribution as

$$\begin{aligned} u(x, y, t)|_w &= -H(x, y, t)U_0'(x)\,, \\ v(x, y, t)|_w &= \dot{H}(x, y, t)\ , \end{aligned} \tag{25}$$

where H denotes the height of the roughness, varying in space and time, $U_0'(x)$ represents the wall-normal derivative of the mean velocity profile at the wall, and $\dot{H}$ is the derivative of the height with respect to time. Since, the shape of the roughness is Gaussian, its height H can be expressed as

$$H(x, y, t) = \varepsilon \exp\left(-\frac{(x-\mu(t))^2}{2\sigma^2}\right)\,, \tag{26}$$

where ε is the height of the curve's peak, σ is its standard deviation or width, and $\mu(t)$ denotes the center of the Gaussian moving with the upper boundary. The control vector is then defined as $\mathbf{g} = [\varepsilon, \sigma]$.

The objective of this optimization process is to extract the most optimal modulation on the casing which will result in a maximum reduction of the average pressure loss across the blade, or in other words, finding the optimal value for $\mathbf{g}$ that minimizes the cost functional

$$\mathcal{J}(\mathbf{q}, \mathbf{g}) = \frac{1}{T}\int_0^T \overline{\Delta p} + \alpha\left(\|u_N(\mathbf{g}, t)\|^2 + \|v_N(\mathbf{g}, t)\|^2\right) dt\,, \tag{27}$$

where $\overline{\Delta p}$ is the spatial average of the total pressure loss around the blade and $u_N(\mathbf{g}, t)$ and $v_N(\mathbf{g}, t)$ are the respective horizontal and vertical boundary conditions imposed on the top of the domain, depending on the control parameters $\mathbf{g}$. The time interval chosen for the optimization is one period of the roughness motion. At $t = 0$ the Gaussian is centered at $x = 0$, and at $t = T$ it is at $x = 1$.

The effect of the unsteady actuation is highlighted in Table 3 by comparing the resulting pressure loss to the uncontrolled setup, resulting in a 1% improvement. The resulting effect on the evolution of the average pressure gradient across the blade, $\overline{\Delta p}$, is also shown in Fig. 11. This figure shows that the presence of the roughness exerts a large influence on the oscillation frequency of the average pressure signal. While in the uncontrolled setup the pressure oscillated with a frequency proportional to the relative difference between the width of the domain and the gap between the blade and the casing surface, in the case with unsteady control the pressure oscillates with the same frequency as the passing of the roughness element. In addition, the actuation increases the amplitude of the oscillations fourfold. The maximum amplitude is reached when the roughness is at $x = 0.35$, shortly before reaching the gap. The lowest amplitude, however, is encountered when the roughness is at $x = 0.77$ before reaching the outlet. Due to the respective frequencies of the pressure signal in the controlled and uncontrolled scenarios, the optima of both curves nearly coincide. Therefore, in order to assess the quantitative

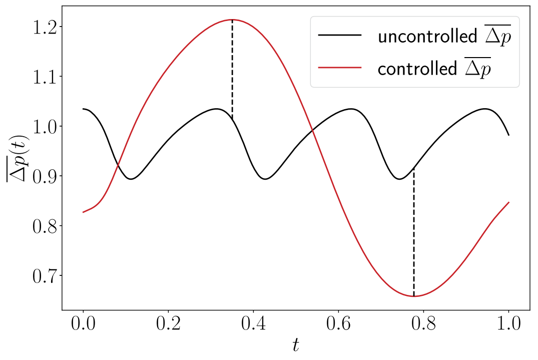


Fig. 11. Evolution of the average pressure loss in time, comparison of the base case (black line), and the case with actuation (red line).

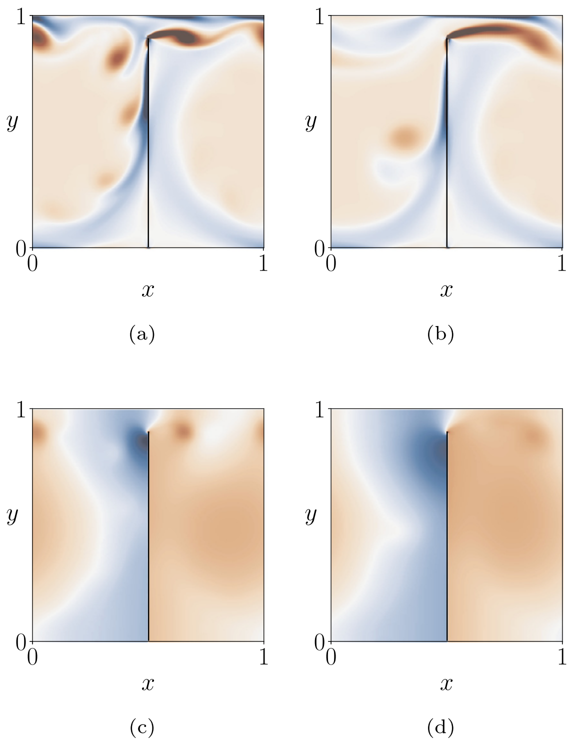
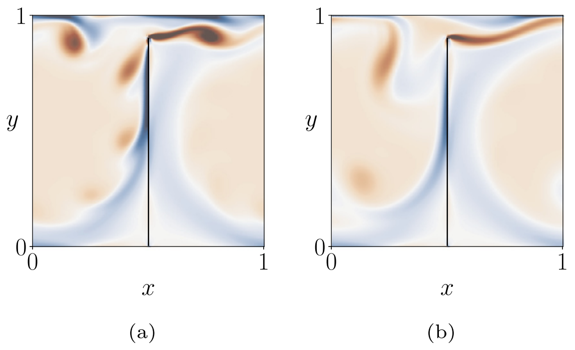


Fig. 12. Vorticity profile and pressure distribution for $t = 0.35$: (a) uncontrolled vorticity, (b) controlled vorticity, (c) uncontrolled pressure, (d) controlled pressure.

differences of the two cases, vorticity and pressure distributions are compared at the maximum and the minimum of the curve describing the pressure loss of the controlled regime, as highlighted by the dashed lines in Fig. 11.

The spatial distributions of pressure and vorticity profiles are displayed in Figs. 12 and 13, at the maximum and the minimum, respectively.

At the time where the average pressure is at its maximum (corresponding to Fig. 12(d)), a large difference between the pressure distribution on the two sides of the blade is noticeable, whereas in Fig. 13(d), corresponding to the point in time where the average pressure is at its minimum, the pressure distribution across the blade is more homogeneous. When comparing the uncontrolled cases in the same two time instances, the main difference is the value of the pressure at the tip of the blade, which appears stronger in the Fig. 12(c) than in Fig. 13(c). However, the pressure distribution along the two sides of the blade seems mostly unaffected. The reason for an overall change in the pressure distribution can be deduced by analyzing the distribution of the vorticity inside the domain. In the uncontrolled case, a vortex is developed on the tip of the blade, which ultimately sheds with a frequency similar to the frequency of the average pressure loss. Due to the periodic boundary conditions, this vortex reenters the domain and creates a series of vortex pairs developing on the suction side of the blade. The shedding process is entirely suppressed by the presence of the roughness. Instead, vorticity is created as the roughness enters and leaves the domain, respectively, leading to a larger vortical structure.



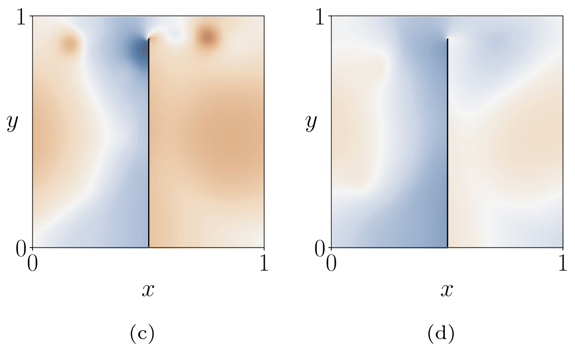


Fig. 13. Vorticity profile and pressure distribution for $t = 0.77$: (a) uncontrolled vorticity, (b) controlled vorticity, (c) uncontrolled pressure, (d) controlled pressure.

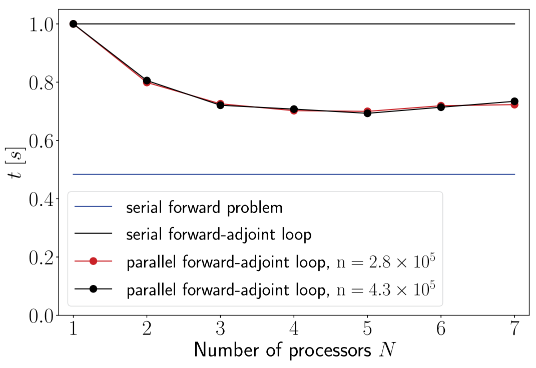


Fig. 14. Performance of the parallel-in-time algorithm (red line) compared to the serial counterpart (black line) reported for a single iteration of the optimization loop, using unsteady control.

The performance of the parallel-in-time algorithm evaluated for one iteration of the optimization algorithm, is shown in Fig. 14. To compare the performance between steady and unsteady actuation, we consider two spatial refinements, similar to the ones used for the previous case, resulting in two state vectors of dimensions $n = 431001$ and $n = 277481$, respectively. As in the previous case, for $N > 4$, the time needed for the communication of the direct solution to all threads exceeds the time saved by the time partitioning and no improvement is accomplished by the parallel-in time procedure. The time to solution reduces towards a maximum of 58% of the total serial time. The magnitude of the gain obtained here is less significant than for the previous case, because of the use of unsteady control. While the resolution and the simultaneity of the forward-adjoint loop is unchanged, the optimality condition (equation (11)) used to compute the gradient, is now time-dependent. Each thread builds the operators needed to solve this equation for its time partition, resulting in additional time. The time required to construct the operators is unavoidable and increases with the time interval chosen for the simulation. It is also the main cause of efficiency loss for the parallel-in-time algorithm with unsteady control. Nonetheless, the additional gain obtained by using the parallel-in-time algorithm is non-negligible, even in the presence of unsteady control.

6. Conclusions

An algorithm for accelerating gradient-based optimization problems has been presented. The algorithm is the extension of the parallel-in-time algorithm for direct-adjoint loops by Skene et al. [45] to the two-dimensional Navier-Stokes equation with immersed boundaries. The pressure and boundary forces are treated by introducing a projection operator to allow the exponential integration of the linear homogeneous adjoint equations using Krylov subspace projection methods. The performance of this method has been tested on two different optimization cases using steady and unsteady control for one gradient evaluation. In both cases the time to solution has been significantly reduced, following a trend consistent with the numerical and theoretical results derived by Skene et al. Better results have been observed for the steady control optimization. In this case, the time required to solve the adjoint-loop converged asymptotically to the time needed to solve the direct equation in serial, suggesting that the computation of the gradient can be obtained with a small additional penalty in overall time. The use of time-dependent control has proven to affect the efficiency of the parallel-in-time procedure. In this case, the gain obtained by using the proposed algorithm is appreciable, but less substantial.

Whilst showing promising results in reducing the computational time, this approach does not address the complications related to the use of gradient-based algorithms in the presence of turbulence or chaotic dynamics.

Recently, Chung & Freund [24] proposed a novel adjoint-based optimization method for chaotic turbulent flows. This method aims to overcome the highly non-convex nature of objective functionals by breaking the time domain into intervals compatible with the chaotic time scales and adding constraints on the intermediate discontinuities generated at the boundaries of each time partition. However, the authors argue that the addition of these intermediate constraints increases overall cost. Making use of our parallel-in-time approach within each such time partition would constitute an interesting option, which will be pursued in a future effort.

CRediT authorship contribution statement

S. Costanzo: Conceptualization, Formal analysis, Methodology, Software, Validation, Visualization, Writing – original draft. **T. Sayadi:** Conceptualization, Supervision, Writing – review & editing. **M. Fosas de Pando:** Formal analysis, Methodology, Software, Writing – review & editing. **P.J. Schmid:** Conceptualization, Methodology, Writing – review & editing. **P. Frey:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

M.F.P. gratefully acknowledges financial support from MINECO/AEI and FEDER/UE through grant DPI2016-75777-R.

References

[1] A. Annaswamy, A. Ghoniem, Active control of combustion instability: theory and practice, IEEE Control Syst. 22 (6) (2002) 37–54.
[2] S. Candel, Combustion dynamics and control: progress and challenges, Proc. Combust. Inst. 29 (1) (2002) 1–28.
[3] O. Pironneau, On optimal design in fluid mechanics, J. Fluid Mech. 64 (1) (1974) 97–110.
[4] A. Jameson, Aerodynamic design via control theory, J. Sci. Comput. 3 (3) (1988) 233–260.
[5] A. Jameson, L. Martinelli, N. Pierce, Optimum aerodynamic design using the Navier-Stokes equations, Theor. Comput. Fluid Dyn. 10 (1) (1998) 213–237.
[6] J. Reuther, A. Jameson, J. Alonso, M. Rimlinger, D. Saunders, Constrained multipoint aerodynamic shape optimization using an adjoint formulation and parallel computers: part 2, J. Aircr. 36 (1) (1999) 61–74.
[7] M. Juniper, Triggering in the horizontal Rijke tube: non-normality, transient growth and bypass transition, J. Fluid Mech. 667 (2010) 272–308.
[8] M. Lemke, J. Reiss, J. Sesterhenn, Adjoint-based analysis of thermoacoustic coupling, in: ICNAAM, 2013, pp. 2163–2166.
[9] S. Schmidt, C. Ilic, V. Schulz, N.R. Gauger, Three-dimensional large-scale aerodynamic shape optimization based on shape calculus, AIAA J. 51 (11) (2013) 2615–2627.
[10] S. Rabin, C. Caulfield, R. Kerswell, Designing a more nonlinearly stable laminar flow via boundary manipulation, J. Fluid Mech. 738 (2014) R1, 1–12.
[11] D. Foures, C. Caulfield, P. Schmid, Optimal mixing in two-dimensional plane Poiseuille flow at finite Peclet number, J. Fluid Mech. 748 (2014) 241–277.
[12] K. Duraisamy, J. Alonso, Adjoint-based techniques for uncertainty quantification in turbulent flows with combustion, in: 42nd AIAA Fluid Dynamics Conference and Exhibit, 2012, pp. 25–28.
[13] K. Braman, T. Oliver, V. Raman, Adjoint-based sensitivity analysis of flames, Combust. Theory Model. 19 (1) (2015) 29–56.
[14] M. Lemke, L. Cai, J. Reiss, H. Pitsch, J. Sesterhenn, Adjoint-based sensitivity analysis of quantities of interest of complex combustion models, Combust. Theory Model. 23 (1) (2019) 180–196.
[15] M. Blanchard, T. Schuller, D. Sipp, P. Schmid, Response analysis of a laminar premixed M-flame to flow perturbations using a linearized compressible Navier-Stokes solver, Phys. Fluids 27 (4) (2015) 043602.
[16] J. Capecelatro, D. Bodony, J. Freund, Adjoint-based sensitivity analysis of ignition in a turbulent reactive shear layer, in: AIAA Sci. Tech. Forum, 2017.

[17] A. Hassan, T. Sayadi, V. LeChenadec, H. Pitsch, A. Attili, Adjoint-based sensitivity analysis of steady char burnout, Combust. Theory Model. (2020).
[18] A. Hassan, T. Sayadi, V. LeChenadec, A. Attili, Sensitivity analysis of an unsteady char particle combustion, Fuel 287 (2021).
[19] A. Fikl, V. Le Chenadec, T. Sayadi, Control and optimization of interfacial flows using adjoint-based techniques, Fluids 5 (2020) 3.
[20] A. Fikl, D.J. Bodony, Adjoint-based interfacial control of viscous drops, J. Fluid Mech. 911 (2021).
[21] N. Kühl, J. Kröger, M. Siebenborn, M. Hinze, T. Rung, Adjoint complement to the volume-of-fluid method for immiscible flows, J. Comput. Phys. 440 (2021) 110411.
[22] J.R. Martins, Aerodynamic design optimization: challenges and perspectives, Comput. Fluids 239 (2022) 105391.
[23] J. Kim, D.J. Bodony, J.B. Freund, Adjoint-based control of loud events in a turbulent jet, J. Fluid Mech. 741 (2014) 28–59.
[24] S.W. Chung, J.B. Freund, An optimization method for chaotic turbulent flow, J. Comput. Phys. 457 (2022) 111077.
[25] R. Vishnampet, D.J. Bodony, J.B. Freund, A practical discrete-adjoint method for high-fidelity compressible turbulence simulations, J. Comput. Phys. 285 (2015) 173–192.
[26] J. Capecelatro, R. Vishnampet, D.J. Bodony, J.B. Freund, Adjoint-based sensitivity analysis of localized ignition in a non-premixed hydrogen-air mixing layer, in: 54th AIAA Aerospace Sciences Meeting, 2016, p. 2153.
[27] J. Capecelatro, D.J. Bodony, J.B. Freund, Adjoint-based sensitivity and ignition threshold mapping in a turbulent mixing layer, Combust. Theory Model. 23 (1) (2019) 147–179.
[28] A. Kord, J. Capecelatro, Optimal perturbations for controlling the growth of a Rayleigh–Taylor instability, J. Fluid Mech. 876 (2019) 150–185.
[29] Q. Wang, R. Hu, P. Blonigan, Least squares shadowing sensitivity analysis of chaotic limit cycle oscillations, J. Comput. Phys. 267 (2014) 210–224.
[30] J. Nievergelt, Parallel methods for integrating ordinary differential equations, Commun. ACM 7 (12) (1964) 731–733.
[31] M.J. Gander, 50 years of time parallel time integration, in: Multiple Shooting and Time Domain Decomposition Methods, Springer, 2015, pp. 69–113.
[32] Y. Maday, M.-K. Riahi, J. Salomon, Parareal in time intermediate targets methods for optimal control problems, in: Control and Optimization with PDE Constraints, Springer, 2013, pp. 79–92.
[33] C. Skene, P. Schmid, Adjoint-based parametric sensitivity analysis for swirling M-flames, J. Fluid Mech. 859 (2019) 516–542.
[34] M. Gander, S. Güttel, A parallel integrator for linear initial-value problems, SIAM J. Sci. Comput. 35 (2013) 123–142.
[35] Y. Cao, S. Li, L. Petzold, R. Serban, Adjoint sensitivity analysis for differential-algebraic equations: the adjoint DAE system and its numerical solution, SIAM J. Sci. Comput. 24 (3) (2003) 1076–1089.
[36] K. Taira, T. Colonius, The immersed boundary method: a projection approach, J. Comput. Phys. 225 (2007) 2118–2137.
[37] M. Fosas de Pando, IBMOS: immersed boundary method for optimization and stability, https://doi.org/10.5281/zenodo.3757783, 2020.
[38] A. Jameson, Aerodynamic Shape Optimization Using the Adjoint Method, Lectures at the Von Karman Institute 2003.
[39] T. Bewley, P. Moin, R. Temam, DNS-based predictive control of turbulence: an optimal benchmark for feedback algorithms, J. Fluid Mech. 447 (2001) 179–225.
[40] M. Wei, J. Freund, A noise-controlled free shear flow, J. Fluid Mech. 546 (2005) 123.
[41] N. Safiran, J. Lotz, U. Naumann, Algorithmic differentiation of numerical methods: second- order adjoint solvers for parameterized systems of nonlinear equations, Proc. Comput. Sci. 80 (2016) 2231–2235.
[42] M. Fosas de Pando, D. Sipp, P. Schmid, Efficient evaluation of the direct and adjoint linearized dynamics from compressible flow solvers, J. Comput. Phys. 231 (23) (2012) 7739–7755.
[43] C. Farhat, J. Cortial, C. Dastillung, H. Bavestrello, Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamic responses, Int. J. Numer. Methods Eng. 67 (2006) 697–724.
[44] M. Gander, S. Güttel, M. Petcu, A nonlinear ParaExp algorithm, in: Int. Conference on Domain Decomposition Methods, Domain Decomposition Methods in Science and Engineering XXIV, 2017, pp. 161–270.
[45] C.S. Skene, M.F. Eggl, P.J. Schmid, A parallel-in-time approach for accelerating direct-adjoint studies, J. Comput. Phys. 429 (2021) 110033.
[46] G.L. Kooij, M.A. Botchev, B.J. Geurts, A block Krylov subspace implementation of the time-parallel Paraexp method and its extension for nonlinear partial differential equations, J. Comput. Appl. Math. 316 (2017) 229–246.
[47] A. Griewank, A. Walther, Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation, ACM Trans. Math. Softw. 26 (1) (2000) 19–45.
[48] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[49] W. Arnoldi, The principle of minimized iterations in the solution of the matrix eigenvalue problem, Q. Appl. Math. 9 (1951) 17–29.
[50] Y. Saad, Analysis of some Krylov subspace approximations to the matrix exponential operator, SIAM J. Numer. Anal. 29 (1) (1992) 209–228.
[51] G.L. Kooij, M.A. Botchev, B.J. Geurts, An exponential time integrator for the incompressible Navier–Stokes equation, SIAM J. Sci. Comput. 40 (3) (2018) B684–B705.
[52] A.J. Chorin, Numerical solution of the Navier-Stokes equations, Math. Comput. 22 (104) (1968) 745–762.
[53] L. Dalcín, R. Paz, M. Storti, MPI for Python, J. Parallel Distrib. Comput. 65 (9) (2005) 1108–1115.
[54] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 contributors, SciPy 1.0: fundamental algorithms for scientific computing in Python, Nat. Methods 17 (2020) 261–272, https://doi.org/10.1038/s41592-019-0686-2.
[55] A.M. Roma, C.S. Peskin, M.J. Berger, An adaptive version of the immersed boundary method, J. Comput. Phys. 153 (2) (1999) 509–534.
[56] J.D. Denton, Loss Mechanisms in Turbomachines, vol. 78897, American Society of Mechanical Engineers, 1993.
[57] I. Jacobi, B. McKeon, Dynamic roughness perturbation of a turbulent boundary layer, J. Fluid Mech. 688 (2011) 258–296.
[58] B. McKeon, A model for 'dynamic' roughness in turbulent channel flow, in: Proceedings of the Summer Program, 2008, pp. 399–410.
[59] A.K.M.F. Hussain, W.C. Reynolds, The mechanics of an organized wave in turbulent shear flow, J. Fluid Mech. 41 (2) (1970) 241–258.