An optimal memory-reduced procedure for calculating adjoints of the instationary Navier-Stokes equations

Michael Hinze
Institute of Numerical Mathematics
Technische Universität Dresden
01062 Dresden
Germany
email: hinze@math.tu-dresden.de

Andrea Walther
Institute of Scientific Computing
Technische Universität Dresden
01062 Dresden
Germany
phone: +49 351 463 34082
fax: +49 351 463 37096
awalther@math.tu-dresden.de

Julia Sternberg
Institute of Scientific Computing
Technische Universität Dresden
01062 Dresden
Germany
phone: +49 351 463 34082
fax: +49 351 463 37096
jstern@math.tu-dresden.de

An optimal memory-reduced procedure for calculating adjoints of the instationary Navier-Stokes equations

Abstract

This paper discusses approximation schemes for adjoints in control of the instationary Navier-Stokes system. It tackles the storage problem arising in the numerical calculation of the appearing adjoint equations by proposing a low-storage approach which utilizes optimal checkpointing. For this purpose, a new proof of optimality is given. This new approach gives so far unknown properties of the optimal checkpointing strategies and thus provides new insights. The optimal checkpointing allows a remarkable memory reduction by accepting a slight increase in run-time caused by repeated forward integrations as illustrated by means of the Navier-Stokes equations. In particular, a memory reduction of two orders of magnitude causes only a slow down factor of 2-3 in run-time.

Keywords: Adjoints, instationary Navier-Stokes equations, binomial checkpointing
AMS (MOS): 49M15, 65K10, 76D55

1 Introduction

Adjoints are the most important tool for computing sensitivities in control problems for the time dependent Navier-Stokes equations. In particular, this is the case when the number of control variables is large and thus forbids approaches based on finite difference methods. To illustrate the role of adjoints for calculating sensitivities let us consider the following abstract optimal control problem: *Find an optimal control $u \in U$ that minimizes the functional*

$$\hat{J}(u) := J(y(u), u), \tag{1}$$

where $y \in Y$ and u are related through the equality constraints

$$G(y, u) = 0 \tag{2}$$

in Z^*. Here and throughout, the superscript * either denotes duals of functions spaces or adjoint operators, where it follows from the context which operation is performed. The superscript $^{-*}$ stands for the corresponding inverse operator. Furthermore, $(U, (\cdot, \cdot)_U)$, $(Y, (\cdot, \cdot)_Y)$ and $(Z, (\cdot, \cdot)_Z)$ denote Hilbert spaces. The functions $J : Y \times U \to \mathbb{R}$ and $G : Y \times U \to Z^*$ are sufficiently smooth mappings. For simplicity, from here onwards we assume that equation (2) admits a unique solution $y(u)$ for every $u \in U$. For the same reason, we assume that $J(y, u) = J_1(y) + J_2(u)$, $G(y, u) = G_1(y) - Bu$ and that $B : U \to Z^*$ is a bounded linear mapping. Let the derivative $G_y(y, u)$ have a bounded inverse for every $(y, u) \in Y \times U$. In Section 2 we will show that control problems for the instationary Navier-Stokes equations fit into this setting.

The first and second derivatives of the functional $\hat{J}$ are the main ingredients of modern optimization algorithms for the numerical solution of (1) – (2). For the action of $\hat{J}'$ at point u in direction δu, we obtain

$$(\hat{J}'(u), \delta u)_U = \langle J_y(y, u), y'(u)\delta u\rangle_{Y^*,Y} + (J_u(y, u), \delta u)_U$$

utilizing the chain rule. The implicit function theorem applied to equation (2) yields

$$y'(u) = -G_y(y, u)^{-1}G_u(y, u),$$

which implies

$$\langle J_y(y, u), y'(u)\delta u\rangle_{Y^*,Y} = (-G_u(y, u)^* G_y(y, u)^{-*} J_y(y, u), \delta u)_U.$$

Introducing the *adjoint variable* $\lambda := -G_y(y, u)^{-*}J_y(y, u) \in Z$ we finally can express the gradient of $\hat{J}$ as

$$\hat{J}'(u) = J_u(y(u), u) + G_u(y(u), u)^*\lambda. \tag{3}$$

In other words, all directional derivatives of the functional $\hat{J}$ can be calculated with the help of the adjoint variable λ, which in turn is obtained by one solve of the *adjoint equation*

$$G_y(y, u)^*\lambda = -J_y(y, u). \tag{4}$$

By similar calculations and transformations we obtain the reduced Hessian, i.e. the second derivative, of the functional $\hat{J}$:

$$\begin{aligned}\hat{J}''(u) = G_u(y(u), u)^* G_y(y(u), u)^{-*} \{&J_{yy}(y(u), u)+\\ &\langle G_{yy}(y(u), u)(\cdot, \cdot), \lambda\rangle_{Z^*,Z}\} \, G_y(y(u), u)^{-1}G_u(y(u), u) + J_{uu}(y(u), u).\end{aligned} \tag{5}$$

From its structure we conclude that the application of $\hat{J}''(u)$ to an element $\delta u \in U$ amounts to the following algorithm

Algorithm I: Reduced Hessian times control

1. Solve $G_y(y(u),u)v = G_u(y(u),u)\delta u$ for $v \in Y$,
2. form right-hand side $g(y,u,\lambda,v) := J_{yy}(y(u),u)v + \langle G_{yy}(y(u),u)(v,\cdot),\lambda\rangle_{Z^*,Z}$,
3. solve $G_y(y(u),u)^*\mu = g(y,u,\lambda,v)$ for $\mu \in Z$,
4. evaluate $\tilde{\mu} := G_u(y(u),u)^*\mu$, and finally,
5. set $\hat{J}''(u)\delta u = \tilde{\mu} + J_{uu}(y(u),u)\delta u$.

One observes, that the adjoint operator $G_y(y(u),u)^*$ is needed twice: One time to evaluate λ defined by equation (4) for a given $u \in U$ and the second time to provide μ in step 3 for applying $\hat{J}''(u)$. Here we note that in the context of inexact Newton methods the evaluation of Hessian times increment plays the central role.

We think of (2) as an abstract realization of the instationary Navier-Stokes equations on the time horizon $[0,T]$ with y denoting the state variables and u serving as control variables. For a given control u, the computation of λ and μ requires knowledge of the state $y(u)$ on the whole time horizon. Therefore, the storage of $y(u)$ forms a serious bottleneck for large time horizons where y represents a 2- or 3-dimensional flow velocity field.

As partial remedy to this storage problem, this paper presents an optimal checkpointing strategy that is applied to compute adjoints for the instationary Navier-Stokes equations. Using the proposed checkpointing technique, a storage reduction of two orders of magnitude is observed, whereas the resulting slow down factor of the adjoint calculation caused by repeated forward integrations of the Navier-Stokes equations lies only between 2 and 3. The main algorithmical tool utilized was revolve which Griewank and the second author developed in [1]. This binomial checkpointing strategy features the adjoint computation within a minimal run time for a given number of checkpoints. In the present paper, a new proof of optimality is given in Theorem 3.5. It uses the new concept of frequency numbers first introduced in [2]. The new technique simplifies the proof significantly and yields in addition so far unknown properties of the optimal checkpointing strategies as for example Theorem 3.4.

As far as the authors know there are only few contributions to control of the instationary Navier-Stokes equations that tackle the storage problem. The technique of equidistant checkpointing and repeated forward integration, also known as windowing, is discussed by Berggren and coauthors in [3, 4]. A detailed comparison of the windowing technique, also called multi-level checkpointing, with the checkpointing strategy proposed in the present paper is contained in [5]. The analysis yields that for a given number of checkpoints and a given upper bound on the number of repeated forward integrations, the checkpointing approach used in this paper allows the adjoint computation for a larger time horizon. Furthermore, it is shown that the time needed for the adjoint computation coincides for both approaches if the windowing is not used recursively. Moreover, it is also proven in [5] that for a given number of checkpoints the time required for the adjoint computation with a recursive windowing exceeds the run time needed for the adjoint computation with the checkpointing method proposed in this paper. Therefore, windowing can be seen as not optimal.

The application of the optimal checkpointing strategy for the Burgers equation is discussed in [6]. This approach has been extended to an adaptive memory and run-time reduced checkpointing strategy by the third author in [2]. Its application for calculating adjoints of the instationary Navier-Stokes system is examined in [7], where the case of a-priory unknown number of time steps with different computational complexity is considered. For this purpose, the extended package a-revolve for adaptive checkpointing is utilized.

For a general analytical framework of first and second order derivatives for control of the instationary Navier-Stokes equations we refer to the work [8] by Kunisch and the first author. For further contributions to (distributed) control of the instationary Navier-Stokes system we refer to Gunzburger and Manservisi [9].

This paper is organized as follows. In Section 2 the instationary Navier-Stokes equations are adapted to the setting introduced at the beginning of Section 1. Appropriate discretizations of the direct and the adjoint PDEs are given. Section 3 introduces reversal schedules including a new proof of optimality for the proposed checkpointing strategies. For this purpose, we introduce for the first time the concept of frequency numbers. They are used to derive new properties of the optimal reversal schedules, which presumably can be exploited in more detail in further studies. Section 4 illustrates the capabilities of the checkpointing algorithm proposed here with respect to memory reduction. The resulting storage requirements and run-times are given and discussed. Finally, Section 5 presents a summary.

2 Adjoints of the Navier Stokes Equations

As model application we now illustrate how derivatives of the reduced functional $\hat{J}$ in control of the instationary Navier-Stokes equations can be realized utilizing the formalism developed in the previous section. We begin with stating an appropriate functional analytic setting (for details see [10, 8]).

2.1 Analytical setting

To define the spaces and operators required for the investigation of the control problem (1) – (2), let $\Omega \subset \mathbb{R}^2$ denote a bounded domain and $[0,T]$ a time horizon. Further, set $Q := \Omega \times (0,T)$. We introduce the solenoidal spaces

$$H = \{v \in C_0^\infty(\Omega)^2 \colon \text{ div } v = 0\}^{-|\cdot|_{L^2}}, V = \{v \in C_0^\infty(\Omega)^2 \colon \text{ div } v = 0\}^{-|\cdot|_{H^1}},$$

with the superscripts denoting closures in the respective norms. For the subsequent considerations it is convenient to introduce the spaces

$$W = \{v \in L^2(V) \colon v_t \in L^2(V^*)\} \quad \text{and} \quad Z := L^2(V) \times H,$$

where W is endowed with the norm

$$|v|_W = |v|_{L^2(V)} + |v_t|_{L^2(V^*)}$$

and V^* denotes the dual space of V. Here $L^2(V)$ is an abbreviation for $L^2(0,T;V)$ and similarly $L^2(V^*) = L^2(0,T;V^*)$. We recall that up to a set of measure zero in $(0,T)$ elements $v \in W$ can be identified with elements in $C([0,T];H)$. By U we denote the Hilbert space of controls which is identified with its dual U^* and we set $\langle \cdot,\cdot \rangle := \langle \cdot,\cdot \rangle_{L^2(V^*),L^2(V)}$. Finally we assume that the cost functional $J \colon (y,u) \in W \times U \to J(y,u) = J_1(y) + J_2(u)$ is bounded from below, weakly lower semi-continuous, twice Fréchet differentiable with locally Lipschitzean second derivative, and radially unbounded in u, i.e. $J(y,u) \to \infty$ as $|u|_U \to \infty$, for every $y \in W$.

Example 2.1. These assumptions are satisfied for cost functionals including tracking type functionals (also with tracking of states at the final time T)

$$J(y,u) = \frac{1}{2} \int\limits_Q |y-z|^2 dx\, dt + \frac{\gamma}{2} \int\limits_\Omega |y(T) - z(T)|^2 dx + \frac{\alpha}{2}\, |u|_U^2, \tag{6}$$

and functionals involving the vorticity of the fluid

$$J(y,u) = \frac{1}{2}\int\limits_Q |\nabla_x \times y(t,\cdot)|^2\, dx\, dt + \frac{\alpha}{2}\, |u|_U^2, \tag{7}$$

where $\alpha, \gamma > 0$ and $z \in W$ are given. Of course, these functionals are even infinitely Fréchet differentiable on $W \times U$. ■

We define the nonlinear mapping

$$G\colon W \times U \to Z^*$$

by

$$G(y,u) = (\tfrac{\partial y}{\partial t} + (y \cdot \nabla)y - \nu\Delta y - Bu, y(0) - y_0),$$

where $B \in \mathcal{L}(U, L^2(V^*))$ denotes the control extension operator, $y_0 \in H$ and $\nu = 1/\mathrm{Re}$ with Re denoting the Reynolds number. In variational form the instationary Navier-Stokes equations can be expressed as: Given $u \in U$ find $y \in W$ such that $y(0) = y_0$ in H and

$$\langle y_t, v\rangle + \langle (y\cdot\nabla)y, v\rangle + \nu(\nabla y, \nabla v)_{L^2(L^2)} = \langle Bu, v\rangle \;\; \forall\, v \in L^2(V), \tag{8}$$

which in turn equivalently can be expressed by

$$G(y,u) = 0.$$

Next we note that for every $u \in U$ equation (8) admits a unique solution $y = y(u)$, see Temam [11]. Utilizing the control-to-state mapping $u \mapsto y(u)$ and defining $Y := W$, the optimal control problem for the Navier-Stokes equations may be formulated as

$$\min\; \hat{J}(u) = J(y(u), u) \;\; \text{subject to} \;\; u \in U, \tag{9}$$

which is exactly of the form (1) – (2). It is proven by Abergel et al. [12] that (9) admits a solution $(y^*(u), u^*) \in W \times U =: X$. Furthermore, the operator $G = (G^1, G^2)\colon X \to Z^*$ is infinitely Fréchet differentiable with Lipschitz continuous first derivative, constant second derivative and vanishing third and higher derivatives, see [10, 8]. As a consequence the differentiability properties of the functional J are carried forward to the functional $\hat{J}$ and it is now clear how the expressions for $\hat{J}'$ and $\hat{J}''$ in (3) and (5), respectively, are to be understood once the actions of the operator $G_y(y,u)^{-1}$ and $G_y(y,u)^{-*}$ are known. To describe these actions for given $J_y(y,u)$, let the function $\lambda \in Z$ be defined by

$$\lambda \;=\; (\lambda^1, \lambda^2) \;=\; -G_y(y,u)^{-*} J_y(y,u),$$

and for $(f, v_0) \in Z^*$ set

$$v := G_y(y,u)^{-1}(f, v_0). \tag{10}$$

From now onwards, we assume that all functions are sufficiently smooth. Precise regularity statements of the variables y, v, and λ can be found in [10, 8].

For numerical considerations it is convenient to formulate the equations (8) in the primitive setting given by

$$\left.\begin{aligned} y_t - \nu\Delta y + (y\cdot\nabla)y + \nabla p &= Bu \text{ in } Q,\\ -\mathrm{div}\; y &= 0 \text{ in } Q,\\ y(x,t) &= 0 \text{ on } \partial\Omega \times (0,T),\\ y(x,0) &= y_0(x) \text{ in } \Omega, \end{aligned}\right\} \tag{11}$$

where p denotes the pressure. Together with the adjoint pressure ξ the adjoint variables λ of (4) satisfies the system

$$\left.\begin{array}{rcl} -\lambda^1_t - \nu\Delta\lambda^1 - (y\cdot\nabla)\lambda^1 + (\nabla y)^t\lambda^1 + \nabla\xi &=& -J^{(t)}_{1_y}(y) \text{ in } Q, \\ -\text{div } \lambda^1 &=& 0 \text{ in } Q, \\ \lambda^1(x,t) &=& 0 \text{ on } \partial\Omega\times(0,T), \\ \lambda^1(x,T) &=& -J^{(T)}_{1_y}(y) \text{ in } \Omega, \end{array}\right\} \quad (12)$$

and $\lambda^2 = \lambda^1(0)$. The superscripts (t), (T) refer to a possible dependence of the representative of the functional J_{1_y} on the time instances (t) and (T), respectively. Finally, together with some pressure h the function v of (10) satisfies

$$\left.\begin{array}{rcl} v_t - \nu\Delta v + (y\cdot\nabla)v + (v\cdot\nabla)y + \nabla h &=& f \text{ in } Q, \\ -\text{div } v &=& 0 \text{ in } Q, \\ v(x,t) &=& 0 \text{ on } \partial\Omega\times(0,T), \\ v(x,0) &=& v_0(x) \text{ in } \Omega. \end{array}\right\} \quad (13)$$

2.2 Discretization

In order to provide a numerical approximation of $\hat{J}'(u)$ for given $u \in U$, the partial differential equations (11) for $y(u)$ and (12) for λ have to be discretized appropriately. For the numerical tests presented in Section 4, Taylor-Hood finite elements are used for spatial discretization, i.e. continuous, piecewise quadratic polynomials for the velocity approximation and continuous, piecewise linear polynomials for the pressure approximation for both y and λ. To discretize w.r.t. time, we fix an equidistant time grid $0 = t_0 < \cdots < t_l = T$, where $t_k := k\Delta t$ and $\Delta t := T/l$.

As time discretization scheme for (11) we apply a semi-implicit Euler-scheme, i.e. implicit w.r.t. diffusive terms, explicit w.r.t. convective terms. The resulting scheme for states y^j with $0 \le j < l$ is given by

$$\begin{array}{c} \frac{y^{j+1}-y^j}{\Delta t} - \nu\Delta y^{j+1} + \nabla p^{j+1} = (Bu)^j - (y^j\nabla y^j) \text{ in } \Omega, \\ -\text{div}\, y^{j+1} = 0 \text{ in } \Omega,\ y^{j+1} = 0 \text{ on } \partial\Omega, \end{array} \quad (14)$$

where $y^0 = y(0)$. To be prepared for Section 3.2 we rewrite this forward integration scheme in the form

$$y^{j+1} = F(y^j, (Bu)^j), \quad (15)$$

where the time step function $F(y^j, (Bu)^j)$ is defined as

$$F(y,z) := \Delta t\,(P - \nu\Delta t S)^{-1}(z - (y\nabla)\,y) + Py. \quad (16)$$

Here, S denotes the Stokes operator and P defines the Leray projection $L^2(\Omega)^{2,3} \to \{v \in L^2(\Omega)^{2,3},\ \text{div } v = 0\}$ [13]. As will be seen in the subsequent Section 3, equation (15) matches exactly the time step definition (22) that forms the basis for the reversal schedules.

The time discretization scheme of the adjoint variables λ in (12) is more involved. Here we take the transpose of the semi-implicit time discretization of equation (13) yielding

$$\begin{array}{c} \frac{v^{j+1}}{\Delta t} - \nu\Delta v^{j+1} + \nabla h^{j+1} = f^j + \frac{v^j}{\Delta t} - (y^j\nabla v^j) - (v^j\nabla y^j) \text{ in } \Omega, \\ -\text{div}\, v^{j+1} = 0 \text{ in } \Omega,\ v^{j+1} = 0 \text{ on } \partial\Omega, \end{array} \quad (17)$$

for $0 \le j < l$, where $v^0 = v(0)$. To derive the time integration scheme for the adjoint variables, it is convenient to reformulate this initial condition in the form

$$\begin{array}{c} \frac{v^0}{\Delta t} - \nu\Delta v^0 + \nabla h^0 = \frac{v_0}{\Delta t} - \nu\Delta v_0 \text{ in } \Omega, \\ -\text{div}\, v^0 = 0 \text{ in } \Omega,\ v^0 = 0 \text{ on } \partial\Omega, \end{array} \quad (18)$$

where we set $h^0 \equiv 0$ and require $v_0 \in V \cap H^2(\Omega)^2$.

Let now $\lambda \equiv \lambda^1$. Formally transposing the scheme (17), (18), we obtain for the adjoint equations and $j = l-k \geq 0$ the integration scheme

$$\begin{aligned} \tfrac{\lambda^{l-k}}{\Delta t} - \nu\Delta\lambda^{l-k} =& \ \tfrac{\lambda^{l-k+1}}{\Delta t} - \nabla\xi^{l-k} - J_{1_y}^{(t_{l-k})}(y^{l-k}) \\ &+ (\lambda^{l-k+1}\nabla y^{l-k}) - (\nabla y^{l-k})^t\lambda^{l-k+1} \ \text{ in } \ \Omega, \\ -\mathrm{div}\,\lambda^{l-k} =& \ 0 \text{ in } \ \Omega, \ \ \lambda^{l-k} \ = \ 0 \text{ on } \ \partial\Omega \end{aligned} \tag{19}$$

where the states y^{l-k} are given by the solution of (14) and $\lambda^i \equiv 0$ for all $i \geq l+1$. We note that the initial state λ^l in this integration scheme satisfies the quasi-Stokes system

$$\begin{aligned} \tfrac{\lambda^l}{\Delta t} - \nu\Delta\lambda^l - \nabla\xi^l =& \ -J_{1_y}^{(t_l)}(y^l, u^l) \text{ in } \Omega, \\ -\mathrm{div}\ \lambda^l = 0 \text{ in } \Omega, \ \lambda^{l-k} \ =& \ \ 0 \text{ on } \ \partial\Omega. \end{aligned}$$

This adjoint integration scheme may be rewritten as

$$\lambda^j = \bar{F}(t_j, y^j, \lambda^{j+1}), \tag{20}$$

where

$$\bar{F}(t, y, z) := \Delta t\,(P - \nu\Delta t S)^{-1}\left(-J_{1_y}^{(t)}(y) + (z\nabla)\,y - (\nabla y)^t\,z\right) + Pz. \tag{21}$$

Next let us briefly discuss the time discretization of $\hat{J}''(u)\delta u$ described in Algorithm I. To begin with we note that

$$\langle G_{yy}(y(u), u)(a, b), (\lambda, \lambda^2)\rangle_{Z^*,Z} = \langle (a\nabla)b + (b\nabla a), \lambda\rangle_{L^2(V^*),L^2(V)},$$

which is independent of y and u, and $G_u(y, u) = (-B, 0)$. To discretize step 1 of Algorithm I we propose to apply scheme (17) – (18) for the computation of v with $v_0 \equiv 0$. In step 2 the numerical results of schemes (14) for y and (19) for λ are utilized to evaluate the right-hand side $g(y, u, \lambda, v)$. Finally, for the numerical computation of μ in step 3, we propose to utilize again scheme (19), where the terms containing the functional J have to be replaced by the corresponding terms of the right-hand side $g(y, u, \lambda, v)$ from step 2. We emphasize that in general both, the state y and the variable v enter into the right-hand side $g(y, u, \lambda, v)$.

Remark 2.2. *The application of the time-discretization scheme* (19) *for the adjoint equations guarantees among other things symmetry of the discretized reduced Hessian, c.f.* (5)*. It also follows immediately from this approach that the states y in the convective terms in* (19) *are taken at the same time instances as the unknown adjoint variables.*

We finally note that $v_0 \in V \cap H^2(\Omega)^2$ is the minimal regularity requirement for proving error estimates for numerical approximation schemes of the Navier-Stokes system, compare [14, 15].

In practical applications of flow control, the storage of the states y (and v) needed for the adjoint calculations forms a serious bottleneck, in particular for large time horizons $[0, T]$. As remedy to this memory requirement problem, in the next section we propose an optimal memory reduced procedure for calculating adjoints which allows a memory reduction of two orders of magnitude for our numerical example while only causing a slow down factor of 2-3 in run-time.

3 Reversal Schedules

In this section we give a general introduction of reversal schedules based on checkpointing strategies. They can be applied for the numerical integration of backward-in-time problems such as (12). To this end, we refer to (2) as realization of a forward PDE and to (4) as realization of an adjoint PDE, respectively. Furthermore, we present a new proof of optimality. Using this new technique, one can derive properties of the optimal reversal schedules that have been unknown so far. They provide new insight and may be used for further theoretical studies.

3.1 Subfunctions, taping and adjoint variables

For calculating an approximation of the state $y(u)$ corresponding to the control u one usually has to evaluate subfunctions F_j, $0 \leq j < l$, called time steps, that act on the state y^j to calculate the subsequent intermediate state y^{j+1} for $0 \leq j < l$ depending on a control action $\bar{u}^j$, i.e.,

$$y^{j+1} \quad = \quad F_j(y^j, \bar{u}^j) \; . \tag{22}$$

In our application we have $F_j(y^j, \bar{u}^j) = F(y^j, (Bu)^j)$ with F from (16). To compute a solution of the adjoint PDE the discretization yield adjoint time steps $\bar{F}_j$ for $l > i \geq 0$ with

$$\lambda^j \quad = \quad \lambda^{j+1} \, F_j'(y^j, \bar{u}^j) \;\equiv\; \bar{F}_j(y^j, \bar{u}^j, \lambda^{j+1}) \; . \tag{23}$$

In our model application, we have $\bar{F}_j(y^j, \bar{u}^j, \lambda^{j+1}) = \bar{F}(t_j, y^j, \lambda^{j+1})$ with $\bar{F}$ given by (21), where the prime denotes differentiation with respect to y. Note that in our application λ^{j+1} does not directly depend on the control action $\bar{u}^j$, which is the case in many control problems for partial differential equations. The evaluation of $\bar{F}_j$ may require some intermediate results calculated during the computation of y^{j+1} from the previous state y^j. Hence, it is supposed that for each $j \in \{0, \ldots, l-1\}$, there exists a preparation step $\hat{F}_j$ that combines the recording of intermediate values onto a data structure called *tape* and the evaluation of F_j. In our application, in addition to the evaluation of F_j the recording step $\hat{F}_j$ performs the storage of y^j and $(Bu)^j$ on disk and/or memory stacks.

Using the recording step $\hat{F}_j$ and the adjoint time step $\bar{F}_j$ the basic way to integrate the adjoint PDE reads as follows.

Algorithm II: Basic approach

Initialization: Set y to the initial value y_0.

Recording: do $j = 0, l-1$
    Perform $y^{j+1} = \hat{F}_j(y^j, \bar{u}^j)$
  end do

Reverse: Initialize the adjoint $\lambda^{l+1} = 0$
  do $j = l, 0, -1$
    Perform $\lambda^j = \bar{F}_j(y^j, \bar{u}^j, \lambda^{j+1})$
  end do

Remark 3.1. *Let us note that the basic approach does not require a-priori discretization of the controls. Rather it requires evaluation of the control actions $\bar{u}$, respectively at the time instances t_j. In our application, we have $\bar{u} = Bu$.*

It is clear that the storage requirement of the basic approach is proportional to the number l of time steps because intermediate data of l time steps are stored during the recording steps. Suppose that we want to calculate the adjoint of a real-world problem with thousand of time steps. Then the memory requirement of the basic approach becomes a problem even if only the intermediate states y_j, $j = 0, \ldots, l$, have to be saved on the tape. For example, for computing 3D flows with unstructured grids one may need easily 10 to 100 MBytes to store one state vector y^j [16]. Therefore, it is reasonable to assume that due to their size, only a very limited number of intermediate states can be kept in memory. They will be used as checkpoints. Applying a checkpointing technique, the required intermediate values are generated piecewise by restarting the evaluation repeatedly from these suitably placed checkpoints, according to requests by the reversal process. Therefore, the calculation of F can be reversed based on a checkpointing strategy, even in such cases where the basic method to calculate adjoints fails due to excessive memory requirement (see e.g. [1, 17]).

3.2 Reversals of Function Evaluations

To derive an optimal reversal strategy, one has to take into account four parameters:

1.) the number l of time steps to be reversed;

2.) the number c of checkpoints that can be accommodated;

3.) the number p of processors that are available; and

4.) the step costs: $\tau_j = TIME(F_j)$, $\hat{\tau}_j = TIME(\hat{F}_j)$, $\bar{\tau}_j = TIME(\bar{F}_j)$.

If the number of time steps l is known a priori, one very popular checkpointing strategy is to distribute the checkpoint equidistantly over the time interval, see e.g. [18]. This technique is also known as windowing, see [4]. However, it was shown in [5] that this checkpointing approach is not optimal. In this paper, we apply well known optimal reversal schedules for serial machines, i.e. $p = 1$, and constant step costs $\tau_j = \tau \in \mathbb{R}$, also known as offline checkpointing. However, we present a new proof of optimality. The optimal offline checkpointing allows an enormous reduction of the memory required to reverse a given evolution in comparison with the basic approach (see e.g. [19, 17, 20]). This achievable memory reduction will be illustrated by the numerical results in Section 4. Even if the step costs $\tau_j = TIME(F_j)$ are not constant it is possible to compute optimal reversal schedules for one processor machines [21]. However in both cases, i.e. constant and non constant step costs, one has to pay for the improvements in the form of a greater temporal complexity because of repeated forward integrations. If a multi-processor can be applied to perform the reversal then optimal parallel reversal schedules ensure that the adjoint values are obtained in minimal wall-clock time [21].

Nevertheless, our motivation was to study the run-time behavior of adjoint calculations using a serial machine. Furthermore, the step costs τ_j are nearly constant. Therefore, we apply the offline checkpointing provided by the routine revolve for constant step costs [1]. To reverse a given time integration, revolve initiates a do-loop similar to the following one:

Algorithm III: Offline checkpointing (reversal schedule)

> Initialization: Reserve space for c checkpoints and set the first one to the initial state.
>
> do $p = l$, 1, -1
>
>> Advance: Starting from the last checkpoint assigned, advance to state y^{p-1}, performing forward time steps without recording of intermediates.

If one or more checkpoints are free, set as many of them as possible to intermediate states along the way.

Reverse: Perform the recording step $y^p = \hat{F}_{p-1}(y^{p-1}, \bar{u}^{p-1})$ and the reverse step $\bar{F}_{p-1}(y^{p-1}, \bar{u}^{p-1}, \lambda^p)$ to calculate the adjoint λ^{p-1}. If state y^{p-1} is stored in a checkpoint, free the checkpoint up for subsequent use.

end do

The question arises, where to place the checkpoints in the Advances of Algorithm III. The strategy implemented in revolve ensures that the initiated reversal process is provably optimal with respect to the run-time increase for a given number of checkpoints. Moreover, it can be shown that the resulting reversal schedules allow a growth of memory and run-time that is only logarithmic in the number l of time steps [19]. One such optimal reversal schedule computed with revolve for ten time steps and three checkpoints is shown in Fig. 1.

Here, the time steps are plotted along the vertical axis. The computing time required for the reversal is represented by the horizontal axis that can be thought of as the computational axis. Each solid horizontal line including the computational axis itself represents a checkpoint. The solid slanted lines represent the Advances of Algorithm III. Here, the setting of a checkpoint is marked with a dot. The corresponding Reverses are visualized by dotted slanted lines.

3.3 Minimal Reversal Cost

Naturally, one wants to analyze the increase in run-time caused by the checkpointing approach in more detail. Usually, the number of checkpoints that fit into the available memory is fixed and the number of time steps to be reversed varies depending on the current problem specification. For given values of the number l of time steps and the number of checkpoints c, let $t(l, c)$ denote the minimal number of time steps F_j that are performed during the execution of a reversal schedule.

To establish an explicit formula for the minimal cost $t(l, c)$ we introduce some characteristic quantities. First, we will examine the number of times one particular time step is evaluated during the execution of a given reversal schedule S:

Definition 3.2 (Repetition Number). *Let a reversal schedule S for the reversal of l time steps be given. The repetition number $r_i(S)$ counts how often the time step F_i is evaluated during the execution of the reversal schedule S. The maximal repetition number $r_{max}(S)$ is defined by*

$$r_{max}(S) = \max_{0 \le i \le l-1} r_i(S).$$

The repetition numbers can be used to group the time steps to be reversed in the following way:

Definition 3.3 (Frequency Number). *Suppose a reversal schedule S for the reversal of l time steps is given. The frequency numbers $m_j(S)$, $0 \le j \le r_{max}(S)$, defined by*

$$m_j(S) \ = \mid \{r_i(S) = j, \ 0 \le i \le l-1\} \mid,$$

count how many time steps are evaluated exactly j times during the execution of the reversal schedule S.

Hence, the operator $|\cdot|$ in Definition 3.3 determines the cardinal number of a given set. To establish an explicit formula for the frequency numbers, a given reversal schedule S will be decomposed into two smaller substructures S^1 and S^2 as illustrated in Fig. 2. As shown in [20, Lemma 12.2], each reversal schedule S can be decomposed into two subschedules S^1 and S^2 without affecting the overall execution cost. This can be done by first storing the intermediate state $y^{\check{l}}$ into a checkpoint. Then the time steps $F_{\check{l}}, ..., F_{l-1}$ are reversed using up to $c-1$ checkpoints at any time with the subschedule S^2. Subsequently, the time steps $F_0, ..., F_{\check{l}-1}$, are reversed using the subschedule S^1. This can be defined by the following binary composition

$$S = S^1 \circ S^2. \tag{24}$$

Since a reversal schedule S for reversing l time steps can also be used to reverse $\tilde{l}$ time steps with $\tilde{l} < l$, we now define $l_{max}(S)$ as the maximal number of time steps that can be reversed with the reversal schedule S and the corresponding $r_{max}(S)$ as defined in Def. 3.2. Using the decomposition (24), one can then easily derive the relations

$$l_{max}(S^1 \circ S^2) = l_{max}(S^1) + l_{max}(S^2), \tag{25}$$

$$r_{max}(S^1 \circ S^2) = \max\left\{r_{max}(S^1) + 1, r_{max}(S^2)\right\}, \tag{26}$$

$$m_j(S) = m_j(S^2) + m_{j-1}(S^1), \tag{27}$$

see, e.g. [2]. We will employ the identities (25) - (27) to derive an explicit formula for the minimal number $t(c, l)$ of time step evaluations that are required for computing the adjoint of an given sequence with l time steps with up to c checkpoints accommodated at any time.

As is proven in [19], the maximal number $l(c, r)$ of time steps, which can be reversed with up to c checkpoints and at most r evaluations of each time step is given by

$$l(c, r) = \beta(c, r) \equiv \binom{c+r}{c} \approx \frac{1}{\sqrt{2\pi}} \left[1 + \frac{r}{c}\right]^c \left[1 + \frac{c}{r}\right]^r \sqrt{\frac{1}{c} + \frac{1}{r}}, \tag{28}$$

using (25). Let $t(S)$ denote the number of time step evaluations that are required by S to compute the adjoint of a sequence with $l_{max}(S)$ time steps. For a reversal schedule S using c checkpoints with

$$\beta(c, r-1) < l_{max}(S) \leq \beta(c, r), \tag{29}$$

and $t(S) = t(c, l_{max}(S))$, i.e., S is optimal, one has obviously

$$r_{max}(S) = r. \tag{30}$$

Using the concept of repetition numbers, now one can prove additionally the following property:

Theorem 3.4 (Explicit Formula for Frequency Numbers). *Assume that l time steps are reversed using Algorithm III with up to c checkpoints accommodated at any time. Let the number of time step evaluations be minimal, i.e. equal to $t(c, l)$. Then, one has*

$$m_j(c, l) = \beta(c-1, j) = \binom{c-1+j}{c-1}, \qquad 0 \leq j < r, \tag{31}$$

where $m_j(c, l)$ denotes the number of time steps evaluated exactly j times for reversing l time steps using up to c checkpoints and the minimal number of time step evaluations $t(c, l)$. The integer r is determined by the equations (29) *and* (30).

Proof: The relation (31) can be proven by induction:

Trivial Case I: $c = 1$.
To achieve the minimal number $t(1, l)$ of time step evaluations, we can explicitly derive the corresponding reversal schedule: The only checkpoint available must be set to the initial state y^0. For the first Reverse action of Algorithm III, the time steps F_j, $0 < j < l-1$ have to be evaluated. To perform the next Reverse action, the time steps F_j, $0 < j < l-2$ have to be evaluated and so on. Hence, for the overall reversal, the first time step F_0 is evaluated $l-1$ times, the second time step F_1 is evaluated $l-2$ times and so on. Therefore, we obtain $m_j(c, l) = 1,\ 0 \le j \le l-1 = r$ since

$$l(1, r-1) = \beta(1, r-1) = r-1 < l \le r = \beta(1, r) = l(c, r).$$

Using the definition of the function $\beta(c, j)$, it follows that

$$m_j(1, l) = 1 = \beta(0, j), \qquad 0 \le j \le r.$$

Trivial Case II: $j = 0$.
As can be seen from Algorithm III, all time steps except the last one, i.e., $\hat{F}_{l-1}$, are evaluated at least one time, since during the reversal schedule only the preparing step $\hat{F}_{l-1}$ and the reverse step $\bar{F}_{l-1}$ but not the original time step F_{l-1} are executed. Therefore, one obtains $m_0(c, l) = 1$ and

$$m_0(c, l) = 1 = \beta(c-1, 0).$$

using once more the definition of the function $\beta(c, j)$. Hence, for the two trivial cases, the identity (31) holds.

Induction step in lexicographical order of (c, j):
The numbers $c > 1$ and $j > 0$ are given. Assume that the assertion (31) is true for all $(\tilde{c}, \tilde{j})$ with $\tilde{j} < j$ or $\tilde{c} = c$ and $\tilde{j} < j$. Now it will be shown that equality (31) is valid for the pair (c, j).

The applied optimal reversal schedule S can be split into two subschedules $S = S^1 \circ S^2$, where the subschedule S^1 uses up to c checkpoints and the subschedule S^2 up to $(c-1)$ checkpoints as explained above. Furthermore, since S is optimal also the subschedules S^1 and S^2 have to be optimal since otherwise S could be improved. Then we have due to the induction assumption that

$$\begin{aligned} m_{j-1}(S^1) &= m_{j-1}(c, \check{l}) = \beta(c-1, j-1), \\ m_j(S^2) &= m_j(c-1, l-\check{l}) = \beta(c-2, j). \end{aligned} \tag{32}$$

Applying the recursive formula (27) for frequency numbers, we obtain

$$\begin{aligned} m_j(S) &= m_j(S^2) + m_{j-1}(S^1) = m_j(c-1, l-\check{l}) + m_{j-1}(c, \check{l}) \\ &= \beta(c-2, j) + \beta(c-1, j-1) = \frac{(c+j-2)!}{(c-2)!\, j!} + \frac{(c+j-2)!}{(c-1)!\,(j-1)!} \\ &= \frac{(c+j-2)!}{(c-2)!\,(j-1)!}\left(\frac{1}{j} + \frac{1}{c-1}\right) = \frac{(c+j-2)!}{(c-2)!\,(j-1)!}\,\frac{(c+j-1)}{j\,(c-1)} \\ &= \frac{(c+j-1)!}{(c-1)!\, j!} = \beta(c-1, j) = m_j(c, l). \end{aligned}$$

Therefore, the relation (31) is proven. ■

Now, we can present a new proof for the explicit formula of $t(c, l)$ that is based on the frequency numbers and much simpler in the argumentation than the original one presented in [1]:

Theorem 3.5 (Minimal Evaluation Cost). *The minimal evaluation cost to reverse l time steps with up to c checkpoints accommodated at any time has the explicit form*

$$t(l,c) = rl - \beta(c+1, r-1), \tag{33}$$

with r being the unique integer satisfying $\beta(c, r-1) < l \leq \beta(c, r)$.

Proof: We will employ the frequency numbers to prove the explicit formula of the minimal cost $t(l,c)$. For given values of l and c, we have

$$t(l,c) = \sum_{j=0}^{r} j\, m_j(c,l) \qquad \text{and} \qquad l = \sum_{j=0}^{r} m_j(c,l). \tag{34}$$

The last equation yields

$$m_r(c,l) = l - \sum_{j=0}^{r-1} m_j(c,l). \tag{35}$$

We derive equation (33) in two steps: First, identity (33) will be proven for the case $l = \beta(c,r)$. Second, we will generalize this result for situations, where the number l of time steps satisfies $l(c, r-1) < l < l(c,r)$.

Step 1: $l = \beta(c,r)$
Using Theorem 3.4 and the identity (35), we have

$$\begin{aligned} m_r(c,l) &= l - \sum_{j=0}^{r-1} m_j(c,l) = l - \sum_{j=0}^{r-1} \beta(c-1, j) = \beta(c,r) - \sum_{j=0}^{r-1} \binom{c-1+j}{c-1} \\ &= \binom{c+r}{c} - \binom{c-1+r}{c} = \binom{c+r-1}{c-1} = \beta(c-1, r). \end{aligned}$$

Using (34), it follows that

$$\begin{aligned} t(l,c) &= \sum_{j=0}^{r} j\, m_j(c,l) = \sum_{j=0}^{r} j\, \beta(c-1,j) = \sum_{j=0}^{r} j \frac{(c+j-1)!}{j!(c-1)!} = \sum_{j=0}^{r} \frac{(c+j-1)!}{(j-1)!(c-1)!} \\ &= \sum_{j=0}^{r} c \frac{(c+j-1)!}{(j-1)!c!} = c \sum_{j=0}^{r} \beta(c, j-1) = c \sum_{j=1}^{r} \binom{c+j-1}{c} = c \binom{c+r-1}{c+1} \\ &= c \frac{(c+r)!}{(c+1)!(r-1)!} = \frac{(c+r)!}{c!(r-1)!} - \frac{(c+r)!}{(c+1)!(r-1)!} = r \binom{c+r}{c} - \binom{c+r}{c+1} \\ &= rl - \beta(c+1, r-1). \end{aligned}$$

Step 2: $\beta(c, r-1) = l(c, r-1) < l < l(c,r) = \beta(c,r)$
For this case, one can derive

$$\begin{aligned} m_r(c,l) &= l - \sum_{j=0}^{r-1} m_j(c,l) = l - \binom{c-1+r}{c} \\ &= l - \beta(c, r-1) + \beta(c,r) - \beta(c,r) = l + \beta(c-1, r) - \beta(c,r). \end{aligned}$$

Using once more (34) yields

$$
\begin{aligned}
t(c,l) &= \sum_{j=0}^{r} j\, m_j(c,l) = \sum_{j=0}^{r-1} j\, \beta(c-1,j) + r\, m_r(c,l) \\
&= \sum_{j=0}^{r-1} j\, \frac{(c+j-1)!}{j!(c-1)!} + r\beta(c-1,r) - r(\beta(c,r)-l) \\
&= \sum_{j=0}^{r-1} \frac{(c+j-1)!}{(j-1)!(c-1)!} + r\frac{(c+r-1)!}{r!(c-1)!} - r(\beta(c,r)-l)
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{j=0}^{r} \frac{(c+j-1)!}{(j-1)!(c-1)!} - r(\beta(c,r)-l) = c\beta(c+1,r-1) - r(\beta(c,r)-l) \\
&= rl + c\frac{(c+r)!}{(c+1)!(r-1)!} - \frac{(c+r)!}{c!(r-1)!} = rl + \frac{(c+r)!}{c!(r-1)!}(\frac{c}{c+1} - 1) \\
&= rl - \beta(c+1,r-1).
\end{aligned}
$$

Hence, equation (33) was proven for any number l of time steps that satisfies the inequality $\beta(c,r-1) < l \leq \beta(c,r)$. ■

Using Theorem 3.5 and Stirling's formula, the equation (28) yields

$$
l \sim \begin{cases} \exp(r)/\sqrt{r} & \text{if} \quad c \sim r \\ c^r & \text{if} \quad c \gg r = const \\ r^c & \text{if} \quad r \gg c = const \end{cases}
$$

as well as the asymptotic behavior

$$
\lim_{l \to \infty} \frac{t(l,c)}{l^{1+c}} = \sqrt[c]{c!} \approx \frac{c}{e}.
$$

Thus we see that binomial checkpointing allows a reduction of the spatial complexity by the factor of size c/l at the expense of an increase in the temporal complements of size $\sqrt[c]{l}$ which can be seen as an attractive way to reduce the enormous memory requirement caused by the basic approach as given by Algorithm II.

4 Numerical results

To illustrate the potential of the checkpointing approach with respect to memory reduction, a cavity flow problem serves as numerical example. The domain is given by the unit square $\Omega := (0,1) \times (0,1)$. We normalized the final time to 1, i.e., $T = 1$, and set $\mathrm{Re} = \frac{1}{\nu} = 10$. The equation

$$
y_0(x) \quad = \quad e \begin{bmatrix} (\cos 2\pi x_1 - 1)\sin 2\pi x_2 \\ -(\cos 2\pi x_2 - 1)\sin 2\pi x_1 \end{bmatrix}
$$

with the Euler number e is used as initial condition. The time-dependent function to be approximated is given by

$$
z(x,t) \quad = \quad \begin{bmatrix} \varphi_{x_2}(x_1,x_2,t) \\ -\varphi_{x_1}(x_1,x_2,t) \end{bmatrix}.
$$

Here, φ is defined through the stream function

$$\varphi(x_1, x_2, t) = \theta(x_1, t)\theta(x_2, t)$$

with $\theta(y, t) = (1 - y)^2(1 - \cos 2k\pi t)$ for $y \in [0, 1]$. Fig. 3 shows the cavity flow at $t = 0.01$ together with the desired flow at $T = 1$. As cost function we chose

$$J(y, u) \quad = \quad \frac{1}{2}\int_0^T \int_\Omega |y - z|^2 dx\, dt + \frac{c}{2}\int_0^T \int_\Omega |u|^2 dx\, dt$$

with $c = 0.01$. In what follows we focus on the numerical evaluation of $\hat{J}'(u)$, where we recall that $\hat{J}(u) = J(y(u), u)$.

For this application, using the recording steps $\hat{F}(y^j, (Bu)^j)$ and the adjoint steps $\bar{F}(t_j, y^j, \lambda^{j+1})$, one may compute the gradient $\hat{J}'(u)$ applying the basic approach as given by Algorithm II. Then, a complete trajectory is stored onto the tape during the forward integration. Subsequently, the tape is read backward during the adjoint calculation. Using a discretization with 2113 velocity nodes and 545 pressure nodes, the storage of the full state information causes a memory requirement of 3.8 MByte for $l = 100$ time steps. If 10.000 time steps have to be performed the memory requirement equals 380 MByte. Note, that only a 2D problem is considered. For computing the solution of a 3D problem the situation gets even worse.

The reversal schedules presented in Section 3 now offer the opportunity to reduce this memory requirement drastically. For that purpose, the routine revolve is employed, which realizes a do-loop like that illustrated by Algorithm III in Section 3. The implementation of revolve is described in detail in [1], to where we refer for detailed implementational issues. Nevertheless, one has to note that for the performed do-loop, in addition to the functions $F(y^j, (Bu)^j)$, $\hat{F}_j(y^j, (Bu)^j)$, and $\bar{F}(t_j, y^j, \lambda^{j+1})$ which are required also for the basic approach to calculate adjoints, only coding of the routines for storing and retrieving a checkpoint is necessary. Therefore, it is usually no problem to incorporate a reversal schedule into the adjoint calculation. The resulting enormous reduction of memory requirement at the cost of a comparably slight increase in run-time is illustrated in the remainder of this section.

The desired gradient $\hat{J}'(u)$ was computed applying revolve and several numbers of checkpoints. Figure 4 shows the observed run-time behavior for $l = 100$ time steps. Here, the vertical axis gives the ratio of the run-time needed to compute $\hat{J}'(u)$ and the run-time to compute $\hat{J}(u)$. The horizontal axis denotes the number of checkpoints used by the reversal schedule.

To compare the achieved results with the basic approach for computing adjoints, one has to note that the run-time for computing $\hat{J}'(u)$ is bounded above by a small constant times the run-time to compute $\hat{J}(u)$. The value of the constant varies between three and five depending on the specific operation counts and memory accesses [20]. As can be seen, the run-time ratio for the checkpointing approach varies between 2 and 5 for a reasonable number of checkpoints. This behavior results in a slow down factor up to 2 compared to the basic approach, where a complete trajectory is stored to compute the adjoint values. That is, using the checkpointing approach the computation of the adjoints is at most twice as slow as the basic approach, where a complete forward trajectory is stored. Nevertheless, the achieved memory requirement can be reduced enormously. Using the same example discretization as mentioned above, namely 2113 velocity nodes and 545 pressure nodes one needs 38 kByte to store one checkpoint. Hence, if the reversal schedule utilize 5 checkpoints, the memory requirement equals 228 kByte for calculating adjoints with reversal schedules. If the basic approach is applied, the memory requirement amount to 3.8 MByte. The dependence of the run-time ratio on the size of the grid as shown in Fig. 4 is remarkable. According to the theory developed in [20] the displayed run-time ratio should be mesh independent. However, so far, there is no explanation for the mesh-dependent behavior and further studies are needed.

The possible memory reduction gets even more impressive if the number of time steps grows. Figure 5 shows the observed run-time behavior for $l = 1000$ time steps. Here, the run-time ratio varies between 2 and 7 for a reasonable number of checkpoints. Compared to the basic approach, where a complete trajectory is stored and read backwards to calculate the adjoints, these run-times represent a slow down factor of up to three. On the other hand, the memory requirement can be reduced drastically in comparison to the basic approach. For example, using 20 checkpoints the adjoint calculation based on the reversal schedule requires 798 kByte for the same discretization as above. Applying the basic approach, the used memory adds up to 38 MByte. Hence, there are two orders of magnitude between the two storage requirements. Therefore, it is shown that reversal schedules enable an immense memory reduction at a slight increase in run-time.

The zig-zagging of the run-time curves in Fig. 5 can be interpreted in the following way: Other processes had to be performed on the machine during the calculation of the gradient is performed. Therefore, the run-time depends also naturally on the current load of the machine. Since the resulting variations vary around a certain run-time ratio if the number of checkpoints exceeds an lower bound one may draw the conclusion that the number of checkpoints is not the main influence on the run-time behavior. Hence above the lower bound, the number of checkpoints can be varied without a big influence on the run-time. The fact is also illustrated by the flat development of the run-time ratios in Fig. 4 if the number of checkpoints exceeds three.

Figure 6 shows two frequency number profiles for 1000 time steps, where $c = 5$ and $c = 10$ checkpoints are available resulting in $r = 8$ in the first case and $r = 4$ in the second case. As we observe, the frequency number profiles for these numbers of checkpoints are quite different. For $c = 5$ checkpoints, we have that $m_8(5, 1000) < m_7(5, 1000)$. This is due to the fact $1000 \ll l_{max}(5, 8) = 1287$. As a consequence, for $c = 5$ the maximal frequency number $\beta(4, 8) = 495$ for 5 checkpoints and 8 repetitions is not attained. Instead, we have $m_8(5, 1000) = 208$ that is given by

$$m_8(5, 1000) = m_8(5, 1000) - (l_{max}(5, 8) - l) = 495 - (1287 - 1000) = 208,$$

see the proof of Theorem 3.5. Hence, only 208 time steps are executed exactly eight times. In the second case, i.e., for $c = 10$ checkpoints and $r = 4$, the inequality $m_4(10, 1000) > m_3(10, 1000)$ is valid. We have $l = 1000 \approx l_{max}(10, 4) = 1001$, which is the maximal number of time steps that can be reversed with $c = 10$ and at most $r = 4$ executions of each time step. Therefore, in this case, almost the maximal frequency number $\beta(9, 4) = 715$ is required, since $l_{max}(10, 4) - l = 1001 - 1000 = 1$. Thus, 714 time steps are executed four times.

All computations were performed on an two-processor personal computer with ADLON 1GHz CPU and 512 MB memory.

5 Summary

This paper in its first part presents recipes for the numerical treatment of reduced gradients and reduced Hessians times increments for cost functionals involving the incompressible Navier-Stokes systems as state equations. Among other things these recipes guarantee symmetry of the discretized reduced Hessian. Their core is a balanced numerical treatment of the Navier-Stokes equations, their derivative and the adjoint of the latter.

For adjoint calculations, one has to provide information computed during the forward integration in reverse order. The basic approach, namely the complete recording of the required information onto one stack, causes an enormous memory requirement.

As remedy, in its second part the paper presents optimal reversal schedules that allow a drastic reduction of the memory requirement at only a comparable slight increase in run-time. In this context, we present a new proof for the optimal complexity yielding also so far unknown properties of the optimal reversal schedules. The resulting memory reduction and run-time behavior is studied for the calculation of adjoints of the incompressible Navier-Stokes equations by applying the checkpointing routine revolve. The achieved results are quite promising. Nevertheless, one has to study further the dependence of the run-time ratios on the size of the discretization.

Acknowledgment

The first author acknowledges support of the Sonderforschungsbereich 609 funded by the Deutsche Forschungsgemeinschaft. Furthermore, we would like to thank the anonymous referees for their valuable comments and hints.

References

[1] Griewank A, Walther A. Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. ACM Trans. Math. Software, 26:19 – 45, 2000.

[2] Sternberg J. Adaptive Umkehrschemata für Schrittfolgen mit nicht-uniformen Kosten. Diploma thesis, Institute of Scientific Computing, TU Dresden, 2002.

[3] Berggren M. Numerical solution of a flow control problem: Vorticity reduction by dynamic boundary action. Siam J. Sci. Comput., 19(3):829–860, 1998.

[4] Berggren M, Glowinski R, Lions JL. A computational approach to controllability issues for flow-related models. I: Pointwise control of the viscous Burgers equation. Int. J. Comput. Fluid Dyn. 7(3):237-252, 1996.

[5] Walther A, Griewank A. Advantages of binomial checkpointing for memory-reduced adjoint calculations. In M. Feistauer et al., editors, Numerical mathematics and advanced applications, pages 834 – 843. Springer, 2004. Proceedings of ENUMATH 2003.

[6] Walther A, Griewank A. Applying the checkpointing routine treeverse to discretizations of burgers' equation. In H.-J. Bungartz, F. Durst, and C. Zenger, editors, High Performance Scientific and Engineering Computing, volume 8 of Lecture Notes in Computational Science and Engineering, pages 13 – 24, Springer, 1999.

[7] Hinze M, Sternberg J. A-Revolve: An adaptive memory and run-time-reduced procedure for calculating adjoints; with an application to the instationary Navier-Stokes system. Optimization Methods and Software, 20(6):645–663, 2005.

[8] Hinze M, Kunisch K. Second order methods for optimal control of time-dependent fluid flow. SIAM J. Control Optim., 40:925–946, 2001.

[9] Gunzburger MD, Manservisi S. Analysis and approximation of the velocity tracking problem for Navier-Stokes flows with distributed control. Siam J. Numer. Anal., 37:1481–1512, 2000.

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

[11] Temam R. Navier-Stokes Equations. North-Holland, 1979.

[12] Abergel F, Temam R. On some Control Problems in Fluid Mechanics. Theoret. Comput. Fluid Dynamics, 1:303–325, 1990.

[13] Constantin P, Foias C. Navier-Stokes Equations. The University of Chicago Press, 1988.

[14] Deckelnick K, Hinze M. Error estimates in space and time for tracking-type control of the instationary Stokes system. International Series on Numerical Mathematics, 143:87–103, 2002.

[15] Heywood JG, Rannacher R. Finite-element approximation of the nonstationary Navier-Stokes problem, I-IV. SIAM J. Numer. Anal., 19:275-311, 23:750-777, 25:489-512, 27:353-384, 1982-1990.

[16] Kaltenbach HJ, Jürgens W, Spille A. Numerische Simulation, Beeinflussung und Eigenmoden-Analyse einer abgelösten Strömung mit Querkomponente. In Ergebnisberichte SFB 557 TP A6, 2001.

[17] Grimm J, Pottier L, Rostaing-Schmidt N. Optimal time and minimum space-time product for reversing a certain class of programs. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 95 – 106. SIAM Philadelphia, 1996.

[18] Heimbach P, Hill C, Giering R. An efficient exact adjoint of the parallel MIT general circulation model, generated via automatic differentiation. Future Generation Computer Systems, 21(8):1356–1371, 2005.

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

[20] Griewank A. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, 2000.

[21] Walther A. Program Reversal Schedules for Single- and Multi-processor Machines. PhD thesis, Institute of Scientific Computing, TU Dresden, 1999.

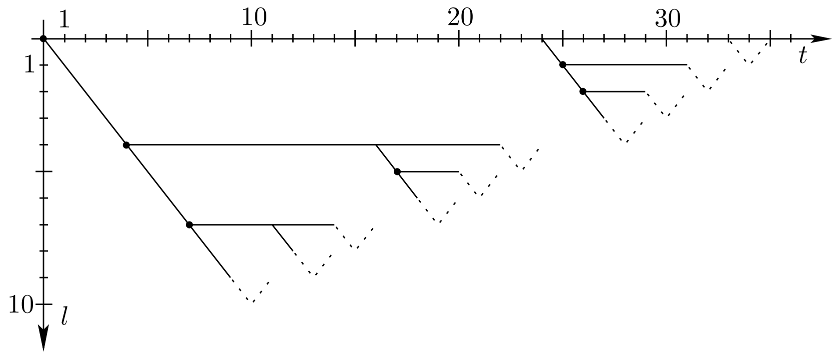

Figure 1: Optimal Reversal Strategy for $l = 10$ and $c = 3$

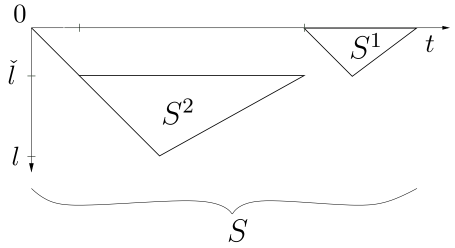

Figure 2: Decomposition of reversal schedule $S = S^1 \circ S^2$

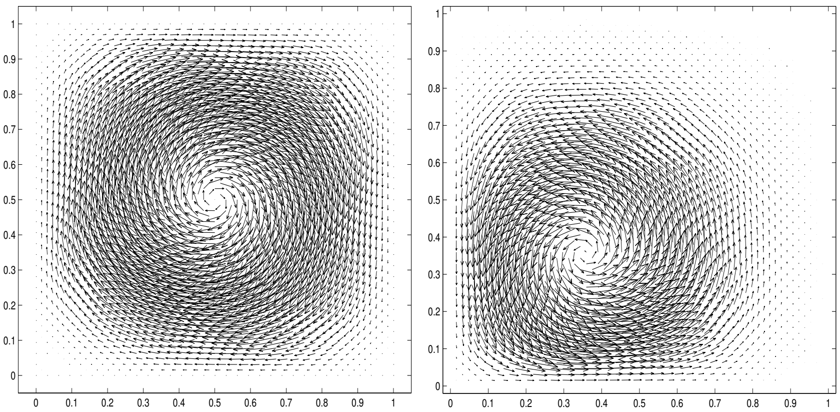

Figure 3: Left: Cavity flow at $t = 0.01$, right: Desired flow at $T = 1$

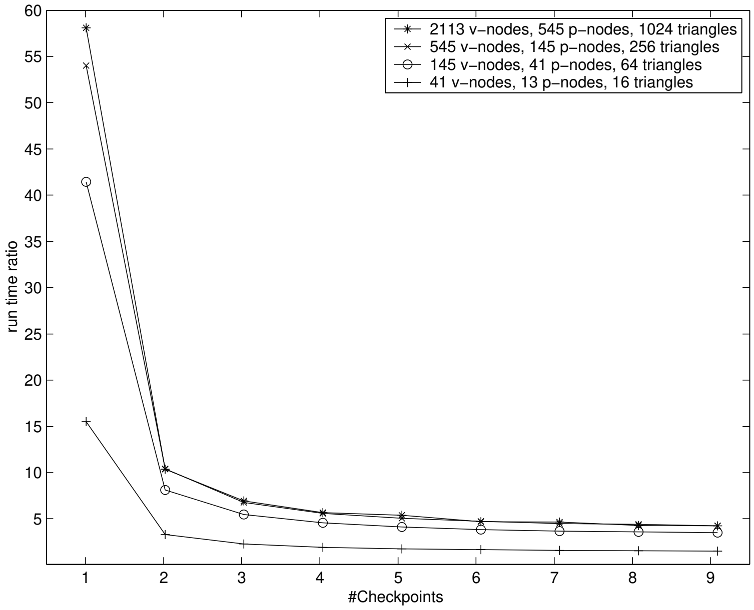

Figure 4: Run-time behavior, 100 Time Steps

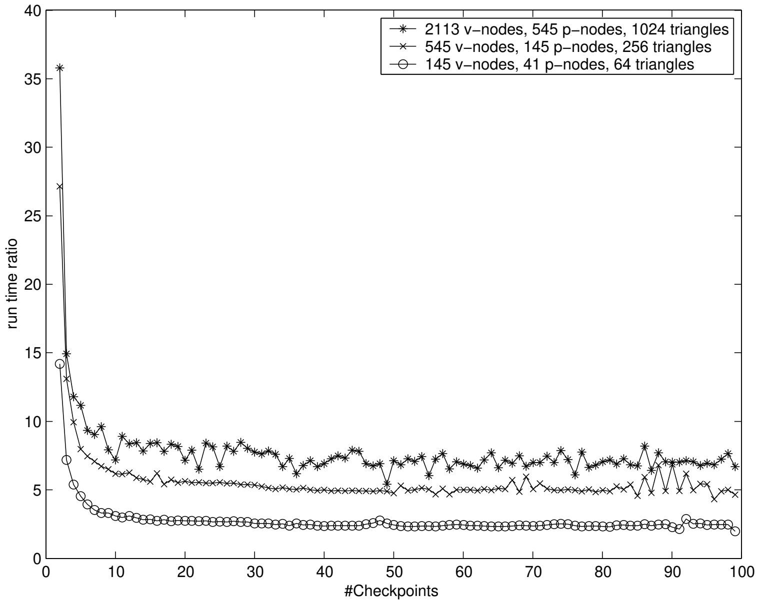

Figure 5: Run-Time behavior, 1000 Time Steps

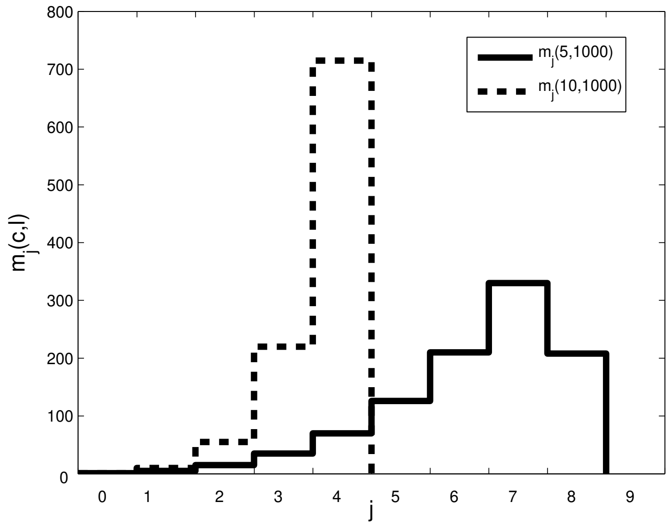

Figure 6: Frequency number profiles for 1000 time steps with 5 and 10 checkpoints