Als Manuskript gedruckt

Technische Universität Dresden
Herausgeber: Der Rektor

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

Michael Hinze and Andrea Walther

MATH-NM-06-2002

June 2002

Contents

1 Introduction **3**

2 Formalism **5**

2.1 Subfunctions, taping and adjoint variables 5

2.2 Reversals of Function Evaluations 7

3 Adjoints of the Navier Stokes Equations **8**

4 Numerical results **13**

5 Conclusions **18**

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

Michael Hinze
Institut für Numerische Mathematik,
Technische Universität Dresden
D–01069 Dresden, Germany,
email: hinze@math.tu-dresden.de

Andrea Walther
Institut für Wissenschaftliches Rechnen,
Technische Universität Dresden
D–01069 Dresden, Germany,
email: awalther@math.tu-dresden.de

Abstract

The paper presents a low-storage approach for calculating numerical approximations of adjoint equations for the instationary Navier-Stokes equations. It utilizes optimal checkpointing. The amounts of memory reduction w.r.t. checkpointing and of the increase in run-time caused by repeated forward integration of the Navier-Stokes equations is reported. It is one result that memory reduction of two orders of magnitude only cause a slow down factor of 2-3 in run-time.

Key words. Optimal control, Navier-Stokes system, optimal checkpointing
AMS(MOS) subject classification. 49M15,49M29,65K10,76D55.

1 Introduction

Adjoints are the most important tool for computing sensitivities in control problems for the time dependent Navier-Stokes equations, in particular 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

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

Here we assume that $J : Y \times U \to \mathbb{R}$ and $G : Y \times U \to Z^*$ are sufficiently smooth mappings and $(U, (\cdot,\cdot)_U)$, $(Y, (\cdot,\cdot)_Y)$ and $(Z, (\cdot,\cdot)_Z)$ denote Hilbert spaces. For simplicity from now onwards we further assume that eq. (2) for every $u \in U$ admits a unique solution $y(u)$ and

that $G_y(y,u)$ for every $(y,u) \in Y \times U$ admits a bounded inverse. In Section 3 we will show that control problems for the instationary Navier-Stokes equations fit into this setting.

For the derivatives of the functional $\hat{J}$ at point u in direction v we obtain, utilizing the chain rule

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

The implicit function theorem applied to eq. (2) yields

$$y'(u) = -G_y(y,u)G_u(y,u)$$

which implies

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

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 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 $J''(u)$ to an element $v \in U$ amounts to

1. Solve $G_y(y(u),u)w = G_u(y(u),u)v$ for $w \in Y$,
2. form $rhs := J_{yy}(y(u),u) + \langle G_{yy}(y(u),u)(w,\cdot), \lambda \rangle_{Z^*,Z}$,
3. solve $G_y(y(u),u)^* \mu = rhs$ for $\mu \in Z$ and finally,
4. evaluate $\tilde{\mu} := G_u(y(u),u)^* \mu$.

Then,

$$\hat{J}''(u)v = \tilde{\mu} + J_{uu}(y(u),u)v.$$

One observes, that the adjoint operator $G_y(y(u),u)^*$ comes into play: To provide μ in step 3. and also λ defined by eq. (4) for given $u \in U$.

Now we think of (2) as the 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. Then, for given control u the computation of λ and μ requires knowledge of the state $y(u)$ on the whole time horizon. In particular for large time horizons with y representing a 2- or 3-dimensional flow velocity field storage forms a serious bottleneck.

As partial remedy to this storage problem in this paper, an optimal checkpointing procedure is developed and is applied to compute adjoints for the instationary Navier-Stokes equations. Its slow down factor in run-time caused by repeated forward integration of the Navier-Stokes equations only lies between 2 and 3 whereas storage reduction of two orders of magnitude is observed. The main algorithmical tool utilized was revolve which Griewank and the second author developed in [5].

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 is discussed by Berggren in [2]. However, his approach is not optimal. Optimal checkpointing for the Burgers equation is discussed in [13]. A general analytical framework for first and second order derivatives for control of the instationary Navier-Stokes equations is presented by Kunisch and the first author in [8].

This paper is organized as follows. Section 2 introduces reversal schedules using a checkpointing strategy. In Section 3 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 4 describes the implementation of our checkpointing strategy and illustrates its capabilities of memory reduction in the calculation of the adjoint states. Moreover, memory requirements and run-time results for both, the basic approach and the checkpointing approach for solving the adjoint PDE are shown. Finally, Section 5 presents some conclusions.

2 Formalism

In the present section we introduce a formal concept for optimal checkpointing in the context of numerical integration of backward-in-time problems. In this context we refer to (2) and (4) as (the realization of a) (forward) PDE and adjoint PDE, respectively.

2.1 Subfunctions, taping and adjoint variables

As is shown in Section 1, adjoint operators are one basic ingredient of the derivatives $\hat{J}'(u)$ and $\hat{J}''(u)$. Their numerically realizations have to be based on appropriate discretizations of adjoint PDEs that have to be integrated backwards in time where the state variables $y(u)$ enter as coefficients and right-hand-sides.

We now adapt the discretization process to the needs of the software tool revolve developed in [5]. To begin with we note that for calculating an approximation of $y(u)$ 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 u^j, i.e.,

$$y^{j+1} \quad = \quad F_j(y^j, u^j) \; . \tag{6}$$

In order to compute a solution of the adjoint PDE the discretization yield adjoint time steps $\bar{F}_j$ for $l > i \geq 0$ with

$$\lambda^j \quad = \quad \lambda^{j+1} \, F_j'(y^j, u^j) \; \equiv \; \bar{F}_j(y^j, u^j, \lambda^{j+1}) \; . \tag{7}$$

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 exist a recording step $\hat{F}_j$ that cause the recording of intermediate values required during the evaluation of the time step F_j onto a data structure called *tape*.

Using the recording step $\hat{F}_j$ and adjoint time step $\bar{F}_j$ the basic way to integrate the adjoint PDE reads as follows. First one evaluates $\hat{F}_j$ for $j = 0, \ldots, l-1$ in order to provide the intermediate values required for calculating an approximation of the solution of the adjoint PDE. Based on this data one evaluates $\bar{F}_j$ for $j = l-1, \ldots, 0$ to compute the desired adjoint information. This basic approach is illustrated by Fig. 1. As can be seen

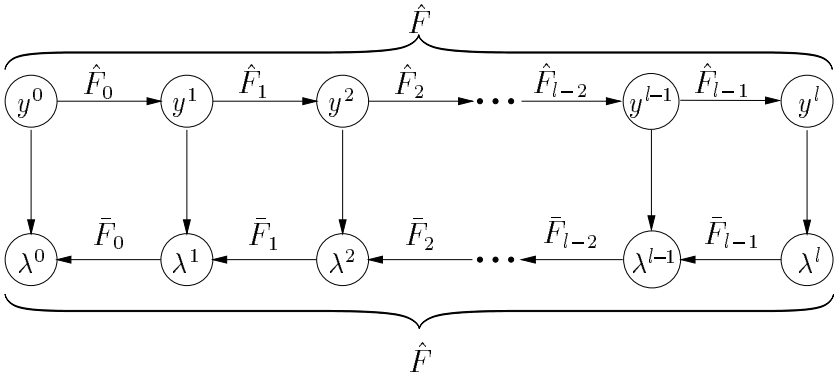

Figure 1: Basic approach to calculate Adjoints

the storage needed for the intermediate results is proportional to the number l of time steps. Therefore, applied to real-world problems with thousand of time steps 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, computing 3D flows with unstructured grids one may need easily 10 to 100 MBytes to store one state vector y^j [9]. For that reason, it will also be assumed that due to their size, only a limited number of intermediate states can be kept in memory. Applying a checkpointing technique, the intermediate values are generated piecewise by restarting the evaluation repeatedly from suitably placed checkpoints, according to requests by the reversal process. Therefore, the checkpoints can be thought of as pointers to nodes representing intermediate states y^j. Using a checkpointing strategy, the calculation of F can be reversed, even in such cases where the basic method to calculate adjoints fails due to excessive memory requirement (see e.g. [6, 5]).

2.2 Reversals of Function Evaluations

Computing the solution of an adjoint PDE can be interpreted as the reversal of a function evaluation because of the backward integration described above. In order to reverse a given evaluation procedure one may employ reversal schedules. These reversal schedules can be formalize as follows.

Definition 2.1 (Reversal Schedule S). Assume the function evaluation to be reversed comprises the time steps F_j, $0, \ldots, l-1$. Let $c \in \mathbb{N}$ checkpoints be available each of which can accommodate one state vector. Then a *reversal schedule* S initializes $j = 0$ and subsequently performs a sequence of basic actions

$$\begin{array}{lcl} A_m & \equiv & \text{Increment } j \text{ by } m \in \{1, \ldots, l-j-1\} \\ D & \equiv & \text{Decrement } l \text{ by 1 if } j = l-1 \\ W_i & \equiv & \text{Copy state } j \text{ to checkpoint } i \in \{1, 2, \ldots, c\} \\ R_i & \equiv & \text{Reset state } j \text{ to checkpoint } i \end{array}$$

until j has been reduced to 0, i.e., the reversal is finished.

This definition of a reversal schedule relies closely on the one described in Chapter 12 of [4]. The action A_m corresponds to a forward sweep, i.e., the time steps $F_k, j \leq k < j+m$, are performed. The execution of a recording step $\hat{F}_j$ and an adjoint time step $\bar{F}_j$ are combined to the basic action D in order to reverse the currently final time step. A checkpoint writing, i.e. the storage of one state vector, is done by W_i. The checkpoint reading, i.e. the recovering of one state vector, is performed by R_i.

Provided $c \geq 1$, it is always possible to find a finite reversal schedule for a given evolution $F_0, \ldots, F_{l-1}$. For example, one may use the reversal schedule S determined by the sequence

$$\begin{array}{rcl} S & \equiv & W_1 + A_{l-1} + D + R_1 + A_{l-2} + D + R_1 + \cdots + A_k + D + R_1 + \\ & & \cdots + A_1 + D + R_1 + D \ , \end{array}$$

where $+$ indicates successive performance and l denotes the given length.

To derive an optimal reversal of the given evaluation procedure, one has to take into account four kinds of parameters, namely:

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

In this paper, we consider well known reversal schedules for serial machines, i.e. $p = 1$, and constant step costs $\tau = \tau_j$. They allow an enormous reduction of the memory required to

reverse a given evolution F in comparison with the basic approach (see e.g. [6, 4]). This achievable memory reduction will be also 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 [12]. 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. For that reason, reversal schedules for multi-processor machines have been developed and implemented [10].

Nevertheless, our motivation was to study the run-time behaviour of adjoint calculations using sequential reversal schedules. Furthermore, the step costs τ_j are nearly constant. Therefore, we apply the reversal schedules provided by the routine revolve [5]. The reversal processes initiated by this program are provably optimal with respect to the run-time increase for a given number of checkpoints. Moreover, it was proven that using revolve a logarithmic increase of memory and run-time can be achieved if the number of time steps to be reversed grows. One optimal reversal schedule computed with revolve for ten time steps and three checkpoints is shown in Fig. 2.

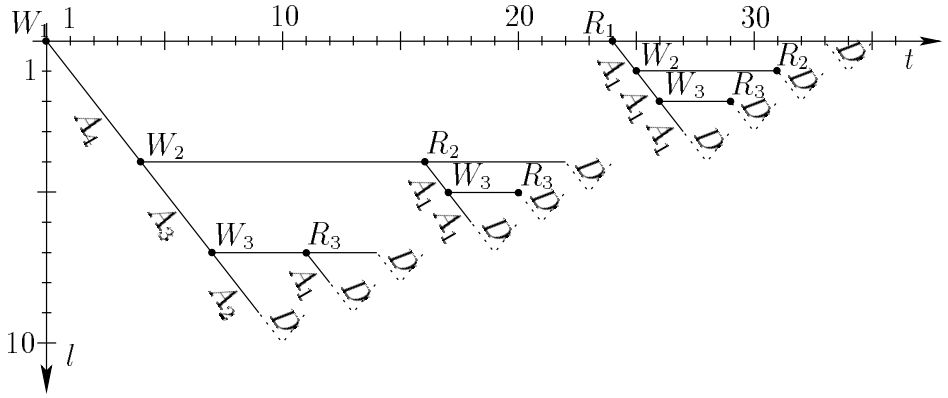

Figure 2: Optimal Distribution of Actions for $l = 10$ and $c = 3$

Here, the time steps are plotted along the vertical axis and the computing time required to reverse F is represented by the horizontal axis. Hence, the horizontal axis can be thought of as the computational axis. Each solid horizontal line including the computational axis itself represents a checkpoint. The writing W_i and reading R_i of a checkpoint is marked with a dot. The solid slanted lines represent the actions A_m, i.e. the execution of time steps F_j without recording. The returning actions D are visualized by dotted slanted lines.

Such optimal reversal schedules generated by revolve will be used in Section 4 to reduce the memory requirement of adjoint computations.

3 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 formal-

ism developed in the previous sections. We begin with stating an appropriate functional analytic setting (for details see [7, 8]).

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, let $Q := \Omega \times (0,T)$. 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. Further we define

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

where W_q^p is endowed with the norm

$$|v|_{W_q^p} = |v|_{L^p(V)} + |v_t|_{L^q(V^*)},$$

abbreviate $W := W_2^2$ and set $\langle \cdot, \cdot \rangle := \langle \cdot, \cdot \rangle_{L^2(V^*),L^2(V)}$, with V^* denoting the dual space of V. Here $L^2(V)$ is an abbreviation for $L^2(0,T;V)$ and similarly $L^2(V^*) = L^2(0,T;V^*)$. Recall that up to a set of measure zero in $(0,T)$ elements $v \in W$ can be identified with elements in $C([0,T];H)$. Further U denotes the Hilbert space of controls which is identified with its dual U^*. We assume that the cost functional $J \colon W \times U \to \mathbb{R} = J_1(y) + J_2(u)$ 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 3.1. These assumptions are satisfied for cost functionals including tracking type functionals (also with tracking of states at the final time T)

$$J(y,u) = \frac{1}{2} \int_Q |y - z|^2 dx\, dt + \frac{\gamma}{2} \int_\Omega |y(T) - z(T)|^2 dx + \frac{\alpha}{2} |u|_U^2, \tag{8}$$

and functionals involving the vorticity of the fluid

$$J(y,u) = \frac{1}{2} \int_Q |\nabla_x \times y(t,\cdot)|^2 \, dx\, dt + \frac{\alpha}{2} |u|_U^2, \tag{9}$$

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 and $y_0 \in H$. 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

$$\langle y_t, v\rangle + \langle (y \cdot \nabla)y, v\rangle + \nu(\nabla y, \nabla v)_{L^2(L^2)} = \langle Bu, v\rangle \ \forall\, v \in L^2(V), \tag{10}$$

which in turn equivalently can be expressed

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

Finally let $Y := W$. Then, the control problem (1),(2) in the setting introduced above reads

$$\min_{(y,u)\in W\times U} J(y, u) \ \text{ subject to } \ G(y, u) = 0 \ \text{ in } Z^*. \tag{11}$$

It is well known, see Temam [11] that for every $u \in U$ (10) admits a unique solution $y(u) \in W$. Therefore, with respect to existence (11) can equivalently be rewritten as

$$\min \hat{J}(u) = J(y(u), u) \ \text{ subject to } \ u \in U, \tag{12}$$

where $y(u) \in W$ satisfies $G(y(u), u) = 0$. It is proved by Abergel *et al.* [1] that (12) admits a solution $(y^*, u^*) \in W \times U =: X$.

In order to express the derivatives of the functional $\hat{J}$ we utilize the linearized Navier-Stokes equations and their adjoints. The corresponding necessities from functional analysis are summarized in the following proposition. Its proof is contained in [7]. For a less general setting see [8].

Proposition 3.2. Let $y \in W$, $v_0 \in H$, and $g \in L^2(V^*)$. Then the system of linearized Navier-Stokes equations

$$A(y)v = (g, v_0) \text{ in } Z^* \Leftrightarrow \left\{ \begin{array}{l} v_t + (v \cdot \nabla)y + (y \cdot \nabla)v - \nu\Delta v = g \quad \text{in } L^2(V^*) \\ v(0) = v_0 \quad \text{in } H, \end{array} \right. \tag{13}$$

admits a unique variational solution $v \in W$. For every $f \in W^*$ the adjoint equation

$$A(y)^* w \ = \ f \quad \text{in } W^*$$

admits a unique variational solution $w = (w^1, w^0) \in Z$. If $f \in L^q(V^*) \cap W^*$ $(1 \le q \le \infty)$, then for every $0 \le \epsilon \le \min\{q - 1, \frac{1}{3}\}$ the function w^1 is an element of $W^2_{1+\epsilon}$ and the variational solution of

$$\left\{ \begin{array}{l} -w^1_t + (\nabla y)^t w^1 - (y \cdot \nabla)w^1 - \nu\Delta w^1 = f \\ w^1(T) = 0. \end{array} \right. \tag{14}$$

Furthermore, it satisfies $w^1(0) = w^0$. The functions v and w^1 satisfy the a-priori estimates

i. $|v|_{L^\infty(H)} + |v|_{L^2(V)} \ \le \ C(|y|_{L^2(V)}) \left\{ |g|_{L^2(V^*)} + |v_0|_H \right\}$,

ii. $|v_t|_{L^2(V^*)} \leq C(|y|_{L^2(V)}, |y|_{L^\infty(H)}) \left\{ |g|_{L^2(V^*)} + |v_0|_H \right\}$,

iii. $|w^1|_{L^2(V)} \leq C(|y|_{L^2(V)}, |y|_{L^\infty(H)}) |f|_{W^*}$.

iv. $|w^1_t|_{L^{1+\epsilon}(V^*)} \leq C(T^{\frac{1-\epsilon}{2(1+\epsilon)}}, T^{\frac{1-3\epsilon}{4(1+\epsilon)}}, |y|_{L^2(V)}, |y|_{L^\infty(H)}) \left\{ |f|_{W^*} + |f|_{L^{1+\epsilon}(V^*)} \right\}$ for all $(0 \leq \epsilon \leq \min\{q-1, \frac{1}{3}\})$.

If in addition $y \in L^\infty(V)$ and $f \in L^2(V^*)$, then $w^1 \in W$ and

v. $|w^1|_{L^2(V)} + |w^1_t|_{L^2(V^*)} \leq C(|y|_{L^\infty(V)}) |f|_{L^2(V^*)}$,

For $y \in W \cap L^\infty(V) \cap L^2(H^2(\Omega)^2)$, $v_0 \in V$ and $g, f \in L^2(H)$ the unique solutions v of (13) and w^1 of (14) are elements of $H^{2,1}(Q)$ and satisfy the a-priori estimates

vi. $|v|_{H^{2,1}(Q)} \leq C(|y|_{L^\infty(V)}, |y|_{L^2(H^2(\Omega)^2)}) \left\{ |g|_{L^2(H)} + |v_0|_V \right\}$

and

vii. $|w^1|_{H^{2,1}(Q)} \leq C(|y|_{L^\infty(V)}, |y|_{L^2(H^2(\Omega)^2)}) |f|_{L^2(H)}$.

The basic ingredients of the derivatives of $\hat{J}$ are the derivatives of the operator G. For the convenience of the reader their properties are stated in

Proposition 3.3. 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. The action of the first two derivatives of G^1 are given by

$$\begin{aligned} \langle G^1_x(x)(w,s), \phi \rangle = \langle w_t, \phi \rangle + \langle (w \cdot \nabla) y, \phi \rangle + \langle (y \cdot \nabla) w, \phi \rangle \\ + \nu (\nabla w, \nabla \phi)_{L^2(L^2)} - \langle Bs, \phi \rangle_{L^2(L^2)}, \end{aligned}$$

where $x = (y,u) \in X$, $(w,s) \in X$ and $\phi \in L^2(V)$, and

$$\langle G^1_{xx}(x)(w,s)(v,r), \phi \rangle = \langle G^1_{yy}(x)(w,v), \phi \rangle = \tag{15}$$

$$\langle (w \cdot \nabla) v, \phi \rangle + \langle (v \cdot \nabla) w, \phi \rangle =: \langle v, H(\phi) w \rangle_{W, W^*}, \tag{16}$$

where $(v,r) \in X$ and $H : L^2(V) \to \mathcal{L}(W, W^*)$.

Note that in this setting the adjoint operator $G_y(y,u)^*$ is given by the adjoint $A(y)^*$ of the linearized Navier-Stokes operator $A(y)$ defined in (13). For sufficiently smooth data the action of $G_y(y,u)^*$ is defined in (14). Further note, that $G_u(y,u) = (B, 0)^t$, where B denotes the control operator.

It is now clear how to express the first and second derivatives of the functional $\hat{J}$ in (3) and (5), respectively. We obtain

$$\hat{J}'(u) = J_u(y,u) + G_u(y,u)^* \lambda = J_u(y,u) + B^* \lambda^1,$$

where for given $J_y(y,u)$ the variable λ is defined through

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

The evaluation of the adjoint operator involved in the application of the reduced Hessian $\hat{J}''(u)$ follows the same lines. For that reason from now onwards we focus on the treatment of the adjoint variables λ.

For numerical considerations it is more convenient to rewrite the instationary Navier-Stokes equations and the adjoints of their linearizations in the primitive setting. For sufficiently smooth data, standard arguments [11] lead to the existence of a distribution p, called pressure which allows to rewrite (10) in the following form: given a control u find the state $y(u)$ and the pressure $p(u)$ such that incompressible Navier-Stokes system

$$\begin{aligned} y_t - \frac{1}{\mathrm{Re}}\Delta y + (y\cdot\nabla)y + \nabla p &= Bu \text{ in } Q, \\ -\mathrm{div}\, y &= 0 \text{ in } Q, \\ y(x,t) &= 0 \text{ on } \partial\Omega\times(0,T), \\ y(x,0) &= y_0 \text{ in } \Omega, \end{aligned}$$

is satisfied. Similarly, the adjoint equations for λ and the adjoint pressure ξ become

$$\left.\begin{aligned} -\lambda^1_t - \tfrac{1}{\mathrm{Re}}\Delta\lambda^1 - (\cdot\nabla)\lambda^1 + (\nabla y)^t\lambda^1 + \nabla\xi &= -J_y^{(t)}(y(u),u) \text{ in } Q, \\ -\mathrm{div}\ \lambda^1 &= 0 \text{ in } Q, \\ \lambda^1(x,t) &= 0 \text{ on } \partial\Omega\times(0,T), \\ \lambda^1(x,T) &= -J_y^{(T)}(y(u),u) \text{ in } \Omega, \end{aligned}\right\} \tag{17}$$

and $\lambda^2 = \lambda^1(0)$. For given u, in order to compute approximations of $y(u)$ and λ the partial differential equations have to be discretized appropriately. For the numerical tests presented in the subsequent section, 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. As time discretization 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 a state y^j with $0 \le j < l$ is given by

$$\begin{aligned} \tfrac{y^{j+1}-y^j}{\Delta t} - \nu\Delta y^{j+1} + \nabla p^{j+1} &= (Bu)^j - (y^j\nabla y^j) \text{ in } \Omega \\ -\mathrm{div}\, y^{j+1} = 0 \text{ in } \Omega,\ y^{j+1} &= 0 \text{ in } \Omega \end{aligned} \tag{18}$$

with $y^0 = y(0)$. In the setting of Section 2.2 this forward integration scheme may be rewritten as

$$y^{j+1} = F(y^j, (Bu)^j), \tag{19}$$

where the time step function $F(y^j,(Bu)^j)$ is given by

$$F(y,z) := \Delta t\left(P - \frac{\Delta t}{\mathrm{Re}}S\right)^{-1}(z - (y\nabla)y) + Py.$$

Here, S denotes the Stokes Operator and P defines the Leray projection $L^2(\Omega)^{2,3} \to \{v \in L^2(\Omega)^{2,3}, \text{ div } v = 0\}$, see [3]. As can be seen, equation (19) matches exactly the time steps (6) that form the basis for the reversal schedules presented in Section 2.

¿From equation (17), one obtains for the adjoint equations and $j = l - k \geq 0$ the integration scheme

$$\begin{aligned} \tfrac{\lambda^{l-k} - \lambda^{l-k+1}}{\Delta t} - \nu \Delta \lambda^{l-k} &= -\nabla \xi^{l-k} - J_y^{(t)}(y^{l-k}, u^{l-k}) \\ +(\lambda^{l-k+1} \nabla y^{l-k}) - &\ (\nabla y^{l-k})^t \lambda^{l-k+1} \ \text{ in } \ \Omega, \\ -\text{div}\, \lambda^{l-k} &= \ 0 \text{ in } \ \Omega, \\ \lambda^l &= \ -J_y^{(T)}(y^l, u^l) \ \text{ in } \ \Omega, \end{aligned} \tag{20}$$

where the states y^{l-k} are given by the solution of (18). This adjoint integration scheme may be rewritten as

$$\lambda^j = \bar{F}(y^j, u^j, \lambda^{j+1}), \tag{21}$$

where in the context of Section 2.2 we have the adjoint time step function

$$\bar{F}(z, y, u) := \Delta t \left(P - \frac{\Delta t}{\text{Re}} S \right)^{-1} \left(-J_y^{(t)}(y, u) + (z\nabla)\, y - (\nabla y)^t\, z \right) + Pz.$$

compare (7). Once more, these adjoint time steps fit exactly for the application of the reversal schedules. Furthermore, one finds that the recording step $\hat{F}(y^j, (Bu)^j)$ consists of the evaluation of $F(y^j, (Bu)^j)$ and the storage of the new state vector y^{j+1}.

Up to now, all presented discretizations and definitions must also be done for the adjoint calculation using the basic approach. Hence, so far nothing differs from the usual way to compute adjoints. The following section presents how the reversal schedules described in Section 2 can be applied to compute adjoints. Moreover, the enormous reduction of memory requirement that can be achieved for the computation of adjoints is presented.

4 Numerical results

A cavity flow problem serves as numerical example. The domain is defined by the unit square $\Omega := (0,1) \times (0,1)$. The final time is normalized to 1, i.e., $T = 1$, and $\text{Re} = 10$. The equation

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

with e = Euler number is used as initial condition. Now, the goal is to approximate the time-dependent desired state given by

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

Here, φ is defined through the stream function

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

with

$$\theta(y, t) = (1 - y)^2(1 - \cos 2k\pi t), \quad y \in [0, 1].$$

In order to measure the quality of the approximation the tracking-type cost function

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

with $c = 0.01$ was chosen. Fig. 3 shows the cavity flow at $t = 0.01$ together with the desired flow at $T = 1$.

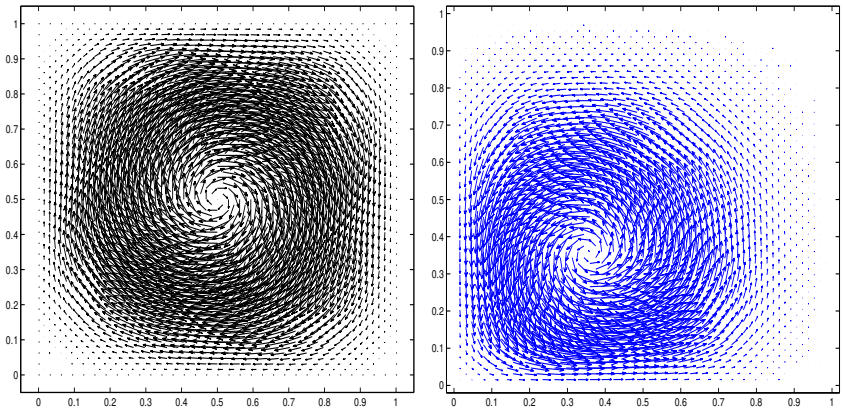

Figure 3: Left: Cavity flow at $t = 0.01$, right: Desired flow at $T = 1$

As mentioned above, using the recording steps $\hat{F}(y^j, (Bu)^j)$ and the adjoint steps $\bar{F}(y^j, u^j, \lambda^{j+1})$ described above, one may compute the gradient $\hat{J}'(u)$ applying the basic approach as illustrated in Fig. 1. With l denoting the number of time steps this basic algorithm reads as follows:

```
init(y)
for j = 1, ..., l
   hF(y,u)
init(lam)
for j = l, ..., 1
   bF(y,u,lam)
```

where hF(y,u) corresponds to $\hat{F}(y^j, (Bu)^j)$ and bF(y,u,lam) to $\bar{F}(y^j, u^j, \lambda^{j+1})$. Using the basic approach, a complete trajectory is stored during the forward integration in the first for-loop. This trajectory is read backward during the adjoint calculation in the second for-loop. 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 worst. The reversal schedules presented in Section 2 offer an opportunity to reduce this memory requirement drastically. For that purpose, the routine revolve may be applied. The corresponding algorithm reads as follows:

```
capo = 0          fine = l          check = -1
init(y)
do
   oldcapo = capo
   whatodo = revolve(check,capo,fine,snaps,info)
   switch(whatodo)
      case advance:  for j = oldcapo, ..., capo
                        F(y,u)
      case takeshot: store(y,ystor,check)
      case firsturn:  hF(y,u)
                     init(lam)
                     bF(y,u,lam)
      case youturn:  hF(y,u)
                     bF(y,u,lam)
      case restore:   restore(y,ystor,check)
      case error:     print(schedule error!)
   end switch
while((whatodo <> terminate) && (whatodo <> error))
```

The first two lines contain initializations. Subsequently, the adjoint calculation is performed within a do-while-loop. Each execution of the loop-body starts with storing the old value of capo and a call of revolve. Here, capo and fine determine the time steps to be reversed currently. The variable snaps denotes the number of available checkpoints and check the number of actual used checkpoints. Finally, info manages the output of additional information about the reversal process. Depending on the actual state of the reversal process, revolve returns a basic action according to the Definition 2.1 of reversal schedules. The return value advance corresponds to the basic action A_m, where $m =$ capo $-$ oldcapo.

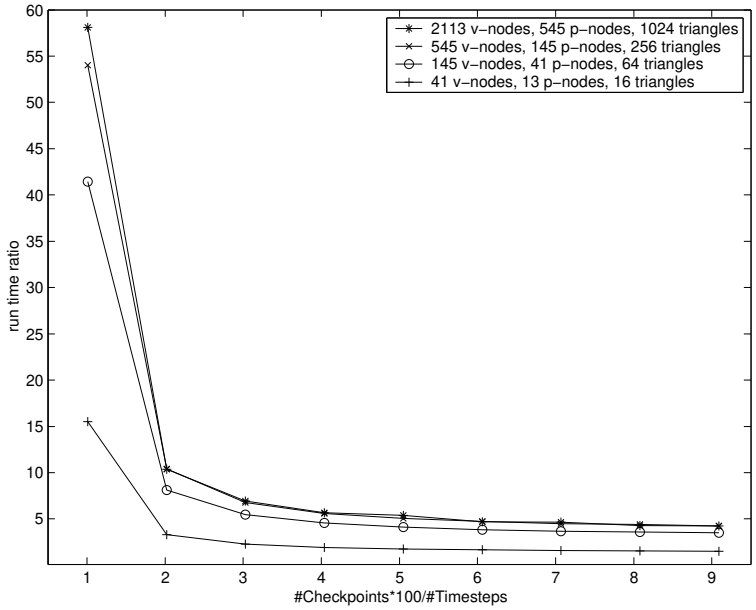

Figure 4: Run-time behaviour, 100 Time Steps

The return value takeshot the basic action W_i whereas restore is equal R_i. For the decrement action D, revolve distinguishes between the first decrement and further decrements. In the first case, revolve returns firsturn so that the initializations of the adjoint variables may be performed. In the latter case, revolve returns youturn.

As can be seen, the do-while-loop is completely independent from the actual problem to be reversed. The functions F(y,u), hF(y,u), and bF(y,u) are required also for the basic approach to calculate adjoints. Hence, in order to apply a reversal schedule to reduce the memory requirement, one has to code only the routines for storing and retrieving a checkpoint in addition to the already written software. 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 the reversal algorithm described above and several numbers of checkpoints. Figure 4 shows the observed run-time behaviour 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 $\hat{J}(u)$. Using the basic approach for computing adjoints, the upper bound for this ratio varies between three and five depending on how to count operations and memory accesses [4]. The horizontal axis denotes the number of checkpoints used by the reversal schedule.

As can be seen, the run-time ratio varies between 2 and 5 for a reasonable number of checkpoints. This behaviour 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. Nevertheless,

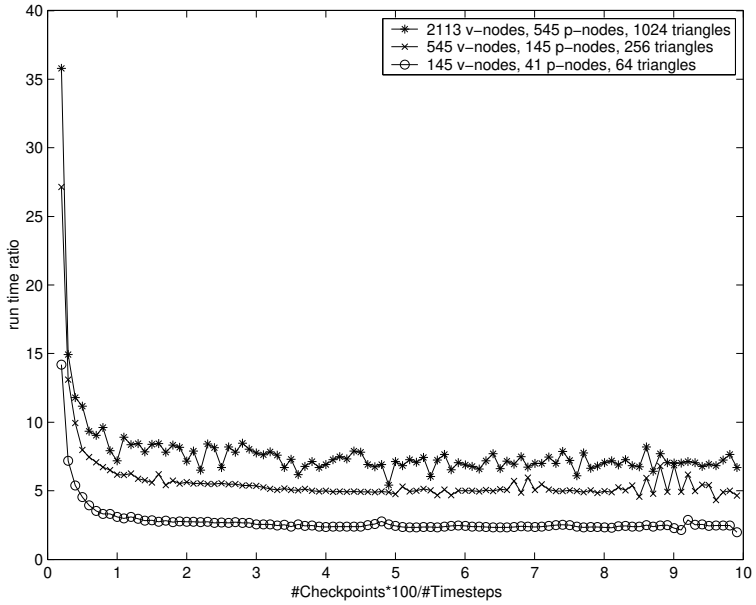

Figure 5: Run-Time behaviour, 1000 Time Steps

the memory requirement is 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 is remarkable. So far, there is no explanation for this behaviour 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 behaviour 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 reread to calculate the adjoints, these run-times represent a slow down factor of up to three. One 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-zaging 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 lie 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 behaviour. 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.

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. Then one can prove the following equation (see, e.g. [5])

$$t(l,c) \;=\; l\,r - \binom{c+r}{c+1}, \tag{22}$$

where r is the unique integer satisfying

$$\binom{c+r-1}{c} \;<\; l \;\leq\; \binom{c+r}{c}$$

Using Stirlings formula one can conclude from eq. (22) that asymptotically

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

Thus we see that 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 seems a very good deal.

5 Conclusions

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.

This paper presents optimal reversal schedules that allow a drastic reduction of the memory requirement at only a comparable slight increase in run-time. The routine revolve implements such optimal reversal schedules. The resulting memory reduction and run-time behaviour is studied for an optimal control problem based on incompressible Navier-Stokes equations. The achieved results are quite promising. Nevertheless, one has to study the dependence of the run-time ratios on the size of the discretization.

References

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

[2] **Berggren, M.** Numerical solution of a flow control problem: Vorticity reduction by dynamic boundary action. *Siam J. Sci. Comput.*, Vol. 19(No. 3):829–860, 1998.

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

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

[5] **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.

[6] **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.

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

[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] **Kaltenbach, H.-J., Jürgens, W.,** & **Spille, A.** Numerische simulation, beeinflussung und eigenmoden-analyse einer abgelösten strömung mit querkomponente. In *Ergebnisberichte SFB 557 TP A6.*

[10] **Lehmann, U.** & **Walther, A.** The implementation and testing of time-minimal and resource-optimal parallel reversal schedules. In *Computational Science - ICCS 2002, Proceedings of the International Conference on Computational Science*, pages 1049 – 1058. Springer, 2002.

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

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

[13] **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, Berlin/Heidelberg, 1999. Springer.