SIAM J. SCI. COMPUT.
Vol. 32, No. 2, pp. 836–854
© 2010 Society for Industrial and Applied Mathematics

# NEW ALGORITHMS FOR OPTIMAL ONLINE CHECKPOINTING*

PHILIPP STUMM† AND ANDREA WALTHER‡

**Abstract.** Frequently, the computation of derivatives for optimizing time-dependent problems is based on the integration of the adjoint differential equation. For this purpose, the knowledge of the complete forward solution may be required. Similar information is needed in the context of a posteriori error estimation with respect to a given functional. In the area of flow control, especially for three dimensional problems, it is usually impossible to keep track of the full forward solution due to the lack of storage capacities. Further, for many problems, adaptive time-stepping procedures are needed toward efficient integration schemes in time. Therefore, standard optimal offline checkpointing strategies are usually not well suited in that framework. In this paper we present two algorithms for an online checkpointing procedure that determines the checkpoint distribution on the fly. We prove that these approaches yield checkpointing distributions that are either optimal or almost optimal with only a small gap to optimality. Numerical results underline the theoretical results.

**Key words.** online checkpointing, calculation of adjoints, program reversals

**AMS subject classifications.** 65Y20, 90C30, 49N90, 68W40

**DOI.** 10.1137/080742439

## 1. Introduction.

In time-dependent flow control as well as in the framework of goal-oriented a posteriori error control, the calculation of adjoint information forms a basic ingredient to generate the required derivatives for the cost functional (see, e.g., [2, 6]). However, the corresponding computations may become extremely tedious if possible at all. This is mainly due to the sheer size of the resulting discretized problem as well as its usually nonlinear character which imposes to a need to keep track of the complete forward solution in order to be able to integrate the corresponding adjoint differential equation backward. This fact still forms a main bottleneck in the overall optimization process despite the ever-growing size of memory devices. For that reason, several checkpointing techniques have been developed [4, 11, 18]. These methods seek an acceptable compromise between memory requirement and runtime increase due to necessary recomputations that cannot be avoided. Hence, it is assumed throughout that a given small number of checkpoints can be kept in memory. These checkpoints store intermediate states to recompute the forward trajectory as required by the adjoint computation.

A general adaptive time-stepping procedure for the state equation for an a priori unknown number of time steps to be performed can be described by

**for** $i = 0, 1, \ldots$
  $x_{i+1} = F_i(x_i)$
  **test stopping criterion**

where $x_i$ denotes the state at time step $i$ and $F_i$ denotes the time integration function for the $i$th time step. If the stopping criterion is fulfilled at time step $l$, the adjoint calculation starts. The adjoint integration scheme is then given by

*Received by the editors December 2, 2009; accepted for publication (in revised form) December 9, 2009; published electronically March 10, 2010.
http://www.siam.org/journals/sisc/32-2/74243.html
†Institute of Scientific Computing, Technische Universität Dresden, Dresden, Germany (Philipp.Stumm@tu-dresden.de).
‡Institut für Mathematik, Universität Paderborn, Paderborn, Germany (Andrea.Walther@uni-paderborn.de).



Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

**for** $i = l, l-1, \dots 0$
$\lambda_i = \bar{F}_i(\lambda_{i+1}, x_i)$

where $\lambda_i$ denotes the adjoint variable corresponding to $x_i$ and $\bar{F}_i$ the adjoint integration operator. If the number of time steps for integrating the considered differential equations is known a priori, one can compute optimal checkpointing schedules in advance to achieve for a given number of checkpoints an optimal, i.e., minimal, runtime increase [4]. This procedure is referred to as offline checkpointing and implemented in the package **revolve** [4]. However, in the context of flow control, the partial differential equations to be solved are usually stiff, and the solution process relies therefore on some adaptive time-stepping procedure. Hence, the number of performed time steps is known only after the complete integration. This fact makes an optimal offline checkpointing impractical. Instead, one may still apply a straightforward checkpointing by placing a checkpoint each time when a certain number of time steps was executed. This approach transforms the uncertainty in the number of time steps to a uncertainty in the number of checkpoints needed. This technique is used, for example, by CVODES [16]. However, when the amount of memory per checkpoint is very high, one certainly wants to determine the number of checkpoints required a priori. For the development of the offline checkpointing strategies, the binomial coefficients $\beta(c, r) \equiv \binom{c+r}{c}$, where $c$ denotes the number of available checkpoints and the integer $r$ is found to be the repetition number, play an important role. Hence, during the adjoint calculation, no time step is executed more than $r$ times.

This paper is an extension of [8], which presented an algorithm for optimal online checkpointing for $\beta(c, 2)$ time steps at maximum without an analysis of the computational complexity. In this paper, we present the theoretical analysis of the online checkpointing algorithm proposed in [8] that distributes a given number of checkpoints during the integration of the state equation to guarantee for a given number of checkpoints a time-optimal adjoint computation. Additionally, we present a new algorithm to generate almost optimal online checkpointing schedules as long as the inequality $\beta(c, 2) < l \leq \beta(c, 3)$ holds. Using these two algorithms a wider range of applications can be covered with optimal or almost optimal online checkpointing. If the number of time steps exceeds $\beta(c, 3)$, one can move ahead with the algorithm presented in [18] that ensures at least a minimal repetition number.

We assume throughout that the time step lengths determined for the forward integration are also used for the adjoint integration, as is often done in the partial differential equation constrained optimization literature [1, 9].

The outline of this paper is as follows. Section 2 presents the online checkpointing strategies together with proofs of optimality and the gap to optimality, respectively. Numerical results are presented in section 3. Finally, we give a summary and a short outlook.

**2. Online checkpointing.** Optimal offline checkpointing schedules that require the minimal runtime for the adjoint calculation for a given amount of memory, i.e., a given number of checkpoints, are well understood if the computational complexities of the time steps are comparable. For example, [4] presents a proof of the following complexity result.

THEOREM 2.1 (optimal offline checkpointing). *Let $t(c, l)$ denote the minimal number of time steps that are evaluated in order to adjoin a given sequence of $l$ time steps storing up to $c$ checkpoints at any time. Then $t(c, l)$ takes the explicit form*

$$t(c, l) = rl - \beta(c + 1, r - 1), \tag{2.1}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

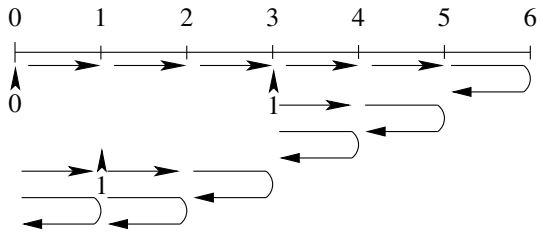



FIG. 2.1. *Optimal offline checkpointing for $c = 2$ and $l = 6$.*

TABLE 2.1
*Maximum time steps for $r = 2, 3$.*

| $c$ | 10 | 20 | 40 | 80 | 160 | 320 |
|---|---|---|---|---|---|---|
| $r = 2$ | 66 | 231 | 861 | 3321 | 13041 | 51681 |
| $r = 3$ | 286 | 1771 | 12341 | 91881 | 708561 | 5564321 |

*where the repetition number $r$ is the unique integer satisfying $\beta(c, r-1) < l \leq \beta(c, r)$.*

*Proof.* See [4]. □

One optimal offline checkpointing for $c = 2$ and $l = 6$ is shown in Figure 2.1. Here, solid lines represent the evaluation of time steps and crooked arrows the evaluation of adjoint steps. The arrow for the adjoint calculation includes a preparing step that is required by automatic differentiation (AD), for example. In case of adjoint calculation based on a continuous formulation, this preparing step may be canceled. The unique integer $r$ denotes the number of times a specific time step is executed during the adjoint computation. For the example shown in Figure 2.1, one has $r = 2$. Hence, no time step is executed more than twice.

As stated before, optimal offline checkpointing can be applied only if the number of time steps $l$ is known in advance. Since this is usually not the case if an adaptive time-stepping procedure is applied, one has to decide on the fly during the first forward integration where to place a checkpoint. Despite this fact, one can derive for a certain range of time steps $l$ provably optimal online checkpoint schemes as shown below. The corresponding range of time steps $l$ is determined by the number of available checkpoints $c$ and the repetition number $r = 2$, i.e., $l \leq \beta(c, 2)$. Subsequently, we present the new algorithm to cover a wider range of time steps $l$, i.e., for $r = 3$ and hence $\beta(c, 2) < l \leq \beta(c, 3)$. Then, the optimality of the checkpointing algorithm can still be proved for a wide range of values of $l$. For the remaining values of $l$, the discrepancy between the reversal cost and the optimal cost can be shown to be only a very small number of time step executions.

Table 2.1 shows the maximum number $\beta(c, 2)$ and $\beta(c, 3)$ of time steps for $r = 2$ and $r = 3$, respectively. For example, if $r = 3$ and $c = 320$, the adjoint of about 5.5 million time steps can be computed for a repetition rate of three. Thus, numerous time-dependent applications can be covered with a moderate number of checkpoints and a repetition number $r = 2, 3$.

**2.1. Optimal online checkpointing for $r = 2$.** First, we derive optimal online checkpointing strategies for an a priori unknown number $l$ of time steps under the assumption that the inequality

$$l \leq \beta(c, 2) = \binom{c+2}{c} = \frac{(c+2)(c+1)}{2} \equiv u_2, \tag{2.2}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

i.e., $r = 2$, holds. For these situations, we prove the minimal complexity of the adjoint computation given by the following online checkpointing procedure:

**Algorithm I:** Optimal Online Checkpointing Algorithm for $r = 2$:

**Start:** Set $i = 0$, $o = c$, $p = c$, $s = 1$
**for** $l = 0, 1, \ldots$
1. Evaluate $x_{l+1} = F_l(x_l)$
2. **If** termination criterion fulfilled, **then**
   start reversal
   **If** $s = 1$, **then**
   Store state $x_l$ in checkpoint $i$
   $i = i + 1$
   **If** $i > o$, **then**
   $i = 1$
3. **If** $l + 1 = p$, **then** $s = 0$
4. **If** $l = p$, **then**
   $p = p + o$, $o = o - 1$, $i = o$
   **If** $o > 0$, **then**
   $s = 1$
   **else**
   $s = 0$
5. **If** $l = p$ **and** $o = -1$, **then**
   error: $l > u_2$

Here, $i$ denotes the number of the checkpoint that stores the next intermediate state, $o$ an offset for the determination of the state at which to place a checkpoint, $p$ the state to be stored in the next checkpoint, and $s$ a flag indicating whether the current state should be stored or not. We assume throughout that the next state $x_{l+1} = F_l(x_l)$ is computed in the time step $l$ while keeping the old state $x_l$ in memory. Hence, $x_{l+1}$ should not overwrite $x_l$.

For a given value of $c$, Algorithm I stores the states $0, \ldots, c-1$ in the checkpoints $0, \ldots, c-1$. Subsequently, the state $c+1$ is copied to the checkpoint $c-1$. Then the states $c+2, \ldots, 2c-1$ are stored in the checkpoints $1, \ldots, c-2$ by overwriting the information already contained in these memory pads. After that, the checkpoint $c-2$ is used to save the state $2c+1$. Now, the states $2c+2, \ldots, 3c-2$ are copied into the checkpoints $1, \ldots, c-3$. This process continues until either the termination criterion is fulfilled or the number of time steps exceeds the upper bound $u_2$. If a reversal is started in step 3, the optimal offline checkpointing derived in the proof of Theorem 2.1 is used for the corresponding adjoint computation. Figure 2.2 illustrates the application of Algorithm I for $c = 4$ and $4 \le l \le 15$ time steps to be reversed. Analyzing the described online checkpointing in more detail, we can prove the following complexity result.

THEOREM 2.2 (optimal online checkpointing for $r = 2$). *Assume that $c$ checkpoints are available. Then the online checkpointing procedure given by Algorithm* I *ensures a time-minimal adjoint computation storing no more than $c$ checkpoints at any time for any number $l$ of time steps if $l$ satisfies the inequality $l \le u_2$.*

*Proof.* We will use the induction principle to show the assertion.

*Case* $c = 1$. It follows that $u_2 = 3$. Hence, we have to check that the reversal process has the optimal complexity for $1 \le l \le 3$. The only checkpoint is used to store the initial state. This yields the following adjoint calculations:

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

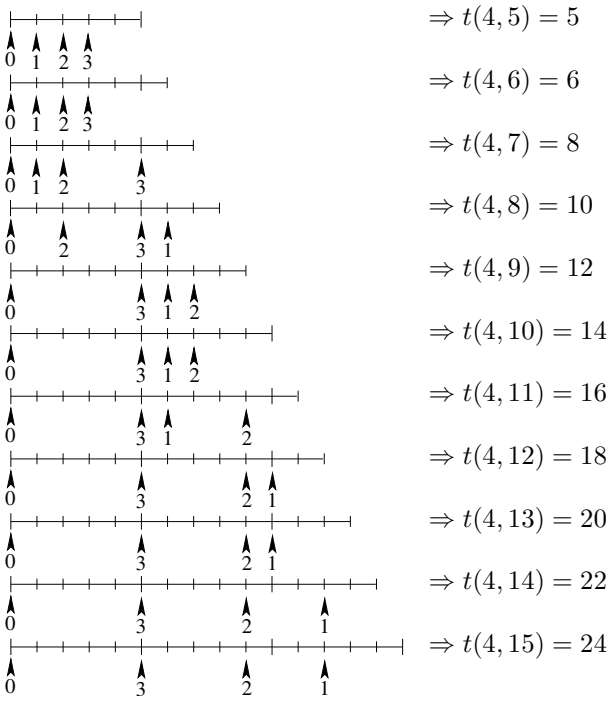
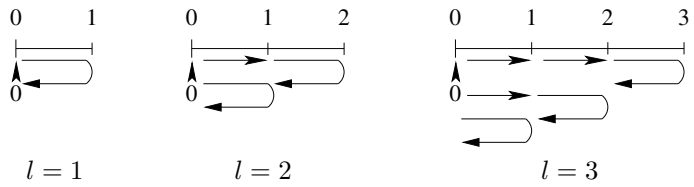



FIG. 2.2. *Checkpoint distributions for* $c = 4$ *and* $l = 5, \ldots, 15$.




Therefore, one obtains that

$$\begin{aligned} l = 1 &\Rightarrow \text{time step evaluations } = 1 = t(1,1), \\ l = 2 &\Rightarrow \text{time step evaluations } = 2 = t(1,2), \\ l = 3 &\Rightarrow \text{time step evaluations } = 4 = t(1,3), \end{aligned}$$

which shows the assertion for $c = 1$

*Induction step* $c - 1 \Rightarrow c$. First, we will consider $l \leq c + 1$. In this case, the states $0, \ldots, l - 1$ are stored in $l - 1 \leq c$ checkpoints. Due to the adjoint calculation considered here, no additional time step evaluation is needed. Hence, only $l$ time steps have to be evaluated to reach the state $l$. On the other hand, we have $l \leq \beta(c, 1) = c + 1$. In this case, Theorem 2.1 yields that the minimal reversal cost equals $t(c, l) = l - \beta(c + 1, 0) + 1 = l$, which proves the assertion for $l \leq c + 1$.

Second, assume that $l = c+2$. Due to Algorithm I, $c+2$ time steps are performed to compute state $l = c + 2$ storing the states $0, \ldots, c - 1$ in $c$ checkpoints. Then, the first adjoint step $\bar{F}_{l-1}$ can be evaluated. Subsequently, the time step $F_{c-1}$ has to be executed to provide the state $c$ for the adjoint step $\bar{F}_c$. Then, all states required for the adjoint calculations are available because of the information stored in the checkpoints. Hence, $l+1$ time steps are evaluated during the reversal. On the other hand, we have $\beta(c, 1) < l \leq \beta(c, 2)$. In this case, Theorem 2.1 yields that the minimal reversal cost

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

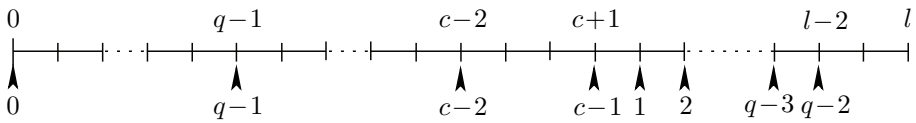



FIG. 2.3. *Checkpoint distribution for* $c+2 < l \leq 2c+1$.

equals

$$t(c,l) = 2l - \beta(c+1,1) + 1 = 2c + 4 - c - 2 + 1 = c + 3 = l + 1,$$

which proves the assertion for $l = c + 2$.

Third, let $c + 2 < l \leq 2c + 1$ be valid. Set $q = l - (c + 1) \leq c$, yielding $1 < q \leq c$. Applying Algorithm I, one obtains that the following states are stored in the checkpoints:

| state | 0 | $q-1$ | $q$ | $\cdots$ | $c-2$ | $c+1$ | $c+2$ | $\cdots$ | $l-2$ |
|---|---|---|---|---|---|---|---|---|---|
| checkpoint | 0 | $q-1$ | $q$ | $\cdots$ | $c-2$ | $c-1$ | 1 | $\cdots$ | $q-2$ |

This situation is illustrated by Figure 2.3. One finds that $l$ time steps are evaluated to compute state $l$. Then, the adjoint calculation from state $l$ down to state $c+1$ can be performed without the evaluation of any time step. Subsequently, the time step $F_{c-2}$ is performed and the state $c-1$ is stored in a checkpoint. Then, the time step $F_{c-1}$ is executed to provide the state $c$. Now, the adjoint calculation can be calculated for the state $c+1$ down to state $q-1$. Finally, the time steps $F_0, \ldots, F_{q-3}$ are executed to store the states $1, \ldots, q-2$ into checkpoints and the adjoint calculation can be finished. Hence, $l + 2 + q - 2 = 2l - c - 1$ time steps are evaluated. On the other hand, we have $\beta(c,1) < l \leq \beta(c,2)$. In this case, Theorem 2.1 yields that the minimal reversal cost equals

$$t(c,l) = 2l - \beta(c+1,1) + 1 = 2l - c - 2 + 1 = 2l - c - 1,$$

which proves the assertion for $c + 2 < l \leq 2c + 1$.

Finally, suppose $2c + 1 < l \leq \beta(c,2)$. Then, the checkpoint 0 stores the initial state 0 and $c + 1$ time steps are performed to reach the state $c + 1$. Subsequently, $c - 1$ checkpoints are used for the reversal of $l - (c + 1)$ time steps. One has

$$l - (c+1) \leq \sum_{i=1}^{c+1} i - (c+1) = \sum_{i=1}^{c} i.$$

Therefore, we can apply the induction assumption that the adjoint of the remaining $l - (c + 1)$ time steps is computed with the minimal number of time step evaluations given by $t(c-1, l-(c+1)) = 2(l-(c+1)) - \beta(c,1) + 1 = 2l - 3c - 2$. Finally, we have to execute the time steps $F_0, \ldots, F_{c-1}$ for storing the intermediate states $1, \ldots, c-1$ into checkpoints and providing the state $c$ for the adjoint calculation which can then be finished without the evaluation of any time step. It follows that the total number of time step evaluations for this adjoint computation is given by $c+1+2l-3c-2+c = 2l - c - 1 = t(c,l)$, and the assertion is proved for $2c + 1 < l \leq \beta(c,2)$. ∎

Hence, it is possible to compute the adjoint of a time step sequence with an a priori unknown length using up to $c$ checkpoints at any time with the optimal, i.e., minimal, runtime provided that the number of time steps does not exceed the upper bound $u_2$. The constant $u_2$ grows quadratically in the number of checkpoints as

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

illustrated also by Table 2.1. Therefore, a moderate number of checkpoints ensures an optimal complexity for the most important applications if a semi-implicit time stepping is applied.

**2.2. Almost optimal online checkpointing for $r = 3$.** Theorem 2.2 guarantees an optimal online checkpointing for $l \leq u_2$. If one detects during the forward integration of the state equation that the number of time steps $l$ exceeds this bound, Algorithm I cannot be applied anymore. In this subsection, we will present a new algorithm, Algorithm II, that can be used as long as $u_2 < l \leq u_3$. Furthermore, we prove that the corresponding adjoint computation requires the minimal runtime for many values $l$ with $\beta(c,2) < l \leq \beta(c,3)$. If this is not the case, the runtime will be almost optimal.

Starting with the result of the optimal online checkpointing for $l = \beta(c,2)$, we will finally reach the optimal checkpoint distribution for $l = \beta(c,3)$ and hence $r = 3$. A general checkpoint distribution for $c$ checkpoints defines the distances $d_j$, as illustrated in Figure 2.4.

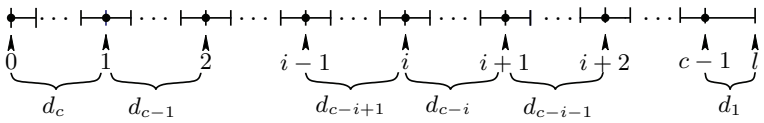



FIG. 2.4. *General checkpoint distribution.*

Let $d_1^s$ denote the distance between the last checkpoint and $l$ and $d_i^s$ the distance between the $(c-i)$th and $(c-i+1)$th checkpoints for $l = u_2$, i.e., at the start of the considered range $[\beta(c,2), \beta(c,3)]$, and $d_i^e$ the corresponding distances for $l = \beta(c,3) = u_3$, i.e., at the end of the considered interval. If $l = u_2$, then one can derive from the optimal online checkpointing that for the distances $d_i^s$ the equation

$$d_1^s = 3\,, \quad d_i^s = i + 2 \quad \text{for} \quad 2 \leq i \leq c$$

is valid. For $l = \beta(c,3)$ the optimal checkpoint distribution is determined by the distances $d_i^e$ given as [4]

$$d_1^e = 4\,, \quad d_2^e = 6\,, \quad d_i^e = d_{i-1}^e + i + 2 \quad \text{for} \quad 2 \leq i \leq c.$$

We assume that the inequalities

$$u_2 = \beta(c,2) < l \leq \beta(c,3) = (c+3)(c+2)(c+1)/6 = u_3 \tag{2.3}$$

hold during the execution of the algorithm presented below. Starting with the optimal distribution for $l = u_2$, new checkpoint distributions will be constructed that are optimal over a special range of $l$. To prove this result, we analyze the number of time step executions that are required to compute the adjoint if a certain checkpoint distribution is given. For a certain checkpoint distribution $C = \{0, d_c, \ldots, \sum_{i=2}^{c} d_i\}$ the number $\hat{t}(c,l)$ of time step executions takes the explicit form

$$\hat{t}(c,l) = \sum_{j=1}^{c} t(j, d_j) + l. \tag{2.4}$$

Optimality, i.e., $\hat{t}(c,l) = t(c,l)$, will not be reached if the next to last state is stored in a checkpoint when the adjoint calculation starts. In this case, $d_1 = 2$ and the

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

checkpoint distribution is not optimal. However, a checkpointing distribution with $d_1 = 2$ is optimal if the adjoint computation does not start but $l$ is increased by one or two as shown by the next proposition.

PROPOSITION 2.1 (almost-optimality condition). *Assume that* (2.3) *holds and let a checkpoint distribution $C$ be given such that $d_1 = 2$ and $\hat{t}(c,l) = t(c,l)+1$. Then this distribution is optimal for $l+1$ and $l+2$.*

*Proof.* Due to (2.3) the repetition number is $r = 3$. Thus it follows from (2.1) that $t(c,l+1) = t(c,l)+3$. With (2.4) one obtains for $l+1$ and the checkpoint distribution $C$ that

$$\hat{t}(c,l+1) = \sum_{j=2}^{c} t(j,d_j) + t(1,3) + (l+1).$$

This yields

$$\begin{aligned}\hat{t}(c,l+1) - t(c,l+1) &= \hat{t}(c,l+1) - t(c,l) - 3\\ &= \hat{t}(c,l+1) - \hat{t}(c,l) - 2\\ &= t(1,3) - t(1,2) + 1 - 2 = 0.\end{aligned}$$

Therefore, the number of recomputations is minimal for $l+1$. For $l+2$ one has

$$\hat{t}(c,l+2) = \sum_{j=2}^{c} t(j,d_j) + t(1,4) + (l+1),$$

and therefore

$$\begin{aligned}\hat{t}(c,l+2) - t(c,l+2) &= \hat{t}(c,l+2) - t(c,l+1) - 3\\ &= \hat{t}(c,l+2) - \hat{t}(c,l+1) - 3\\ &= t(1,4) - t(1,3) + 1 - 3 = 0,\end{aligned}$$

proving the minimal number of recomputations. ∎

Thus, if $\hat{t}(c,l) = t(c,l)+1$ holds for a checkpoint distribution for $l$ time steps, then this distribution is optimal if $l$ is increased by one or two. Hence, it is a reasonable strategy to store an intermediate state in a checkpoint in every third time step. Furthermore, it follows that the next to last state should not be stored in a checkpoint, since otherwise the runtime for the adjoint calculation would increase considerably. Therefore, we decide to store the corresponding state only in a checkpoint if the termination criterion is not reached in the next time step.

Next, we analyze the situation when a checkpoint is replaced for the first time when $l > u_2$. For this purpose, we need the following proposition.

PROPOSITION 2.2. *The checkpoint distribution for $l = u_2$ time steps is also optimal for $u_2 + 1$ time steps.*

*Proof.* First we show that $t(c,u_2+1) = t(c,u_2) + 3$. One has for the repetition number $r$ that $r = 2$ for $l = u_2$ and $r = 3$ if $l = u_2 + 1$. Furthermore, one has

$$t(c,u_2) = 2u_2 - \beta(c+1,1), \qquad t(c,u_2+1) = 3(u_2+1) - \beta(c+1,2).$$

This implies

$$\begin{aligned}t(c,u_2+1) - t(c,u_2) &= 3(u_2+1) - \beta(c+1,2) - 2u_2 + \beta(c+1,1)\\ &= u_2 - \frac{(c+3)(c+2)}{2} + (c+2) + 3\\ &= \frac{(c+2)(c+1)}{2} - \frac{(c+3)(c+2)}{2} + (c+2) + 3 = 3.\end{aligned}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Since $\hat{t}(c, u_2) = t(c, u_2)$, we obtain

$$\begin{aligned}\hat{t}(c, u_2 + 1) - t(c, u_2 + 1) &= \hat{t}(c, u_2 + 1) - t(c, u_2) - 3 \\ &= \hat{t}(c, u_2 + 1) - \hat{t}(c, u_2) - 3 \\ &= t(1, 4) - t(1, 3) + 1 - 3 = 0,\end{aligned}$$

proving the optimality. ∎

Hence, if the value of $r$ increases from two to three because of an increase in the number of time steps, the content of a checkpoint is replaced for the first time if $l$ reaches $\beta(c, 2) + 2$. The following theorem describes the replacement condition in more detail.

THEOREM 2.3 (replacement condition). *Let a checkpoint distribution $C$ be given such that $\hat{t}(c, l) = t(c, l) + 1$. Applying one of the rules*

1. *the content of a checkpoint $i = 1, \ldots, c - 2$ is replaced by $x_{l+1}$ in the time integration from $x_l$ to $x_{l+3}$ if*

$$d_{c-i+1} + d_{c-i} \leq d^e_{c-i+1} \quad \text{and} \quad d_j > d^s_j \quad \text{for } j = 2, \ldots, c - i - 1$$

*holds, or*

2. *the content of the last checkpoint $c - 1$ is replaced by $x_{l+1}$ if $d_2 = 3$*

*yields $\hat{t}(c, l + 3) = t(c, l + 3) + 1$.*

*Proof.* We show for both cases that $\hat{t}(c, l+3) = \hat{t}(c, l)+9$ holds because this yields $\hat{t}(c, l + 3) = t(c, l + 3) + 1$.

First, we consider rule 1. We denote by $r(c_i, d_i)$ the repetition number for a distance $d_i$ and $c_i$ checkpoints such that $\beta(c_i, r(c_i, d_i) - 1) < d_i \leq \beta(c_i, r(c_i, d_i))$. When storing the state $x_{l+1}$ in checkpoint $i$, the old and new checkpoint distributions are given as shown in Figure 2.5.

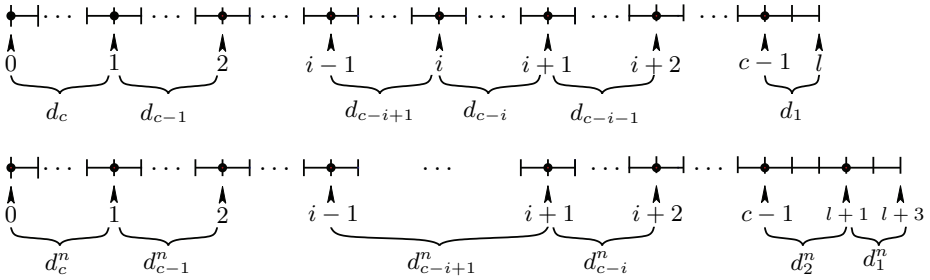



FIG. 2.5. *Replacing a checkpoint.*

For the distances between the checkpoints, the equations

$$\begin{array}{lrcll}
\text{(A1)} & d^n_j & = & d_j & \text{for } j = c - i + 2, \ldots, c, \\
\text{(A2)} & d^n_j & = & d_{j-1} & \text{for } j = 3, \ldots, c - i - 1, \\
\text{(A3)} & d^n_{c-i+1} & = & d_{c-i+1} + d_{c-i}, & \\
\text{(A4)} & d^n_2 & = & 3, & \\
\text{(A5)} & d^n_1 & = & 2 &
\end{array}$$

are valid. Furthermore, one has $r(j, d^s_j) = 1$ and $r(j, d^e_j) = 2$ [4]. Since $d^s_j \leq d^n_j \leq d^e_j$, the repetition number $r(j, d^n_j)$ is either one or two. Now, we determine $t(j, d^n_j)$ for $j = 1, \ldots, c$ by considering each case (A1)–(A5).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

First, we consider (A1), i.e., $j = c - i + 2, \dots, c$. Here, the distances and the number of checkpoints in the old and new checkpoint distribution coincide. This yields that $t(j, d_j^n) = t(j, d_j)$ for $j = c - i + 2, \dots, c$.

Second, we consider (A2), i.e., $j = 3, \dots, c - i + 1$. The equality $d_j^n = d_{j-1}$ holds, and there exists one additional checkpoint for the distances $d_j^n$. Since $d_j > d_j^s$, the repetition number $r(j, d_j^n)$ equals 2. Applying Theorem 2.1, one obtains

$$t(j, d_j^n) = 2d_j^n - (j+2) \quad \text{and} \quad t(j-1, d_{j-1}) = 2d_{j-1} - (j+1) = 2d_j^n - (j+1).$$

This yields

$$t(j, d_j^n) = t(j-1, d_{j-1}) - 1.$$

Third, we consider the case (A3), i.e., $j = c - i + 1$. Hence one has $d_{c-i+1}^n = d_{c-i+1} + d_{c-i}$. We show that the equation

$$t(c-i+1, d_{c-i+1}^n) = t(c-i+1, d_{c-i+1}) + t(c-i, d_{c-i}) + (c-i+2)$$

is valid. Let $r_1 = r(c-i, d_{c-i})$ be the repetition number for $d_{c-i}$ and $r_2 = r(c-i+1, d_{c-i+1})$ be the repetition number for $d_{c-i+1}$, respectively. Then one has

$$r_1 = \begin{cases} 1 & \text{if } d_{c-i} = d_{c-i}^s, \\ 2 & \text{if } d_{c-i} > d_{c-i}^s, \end{cases}$$

and, respectively, we obtain

$$r_2 = \begin{cases} 1 & \text{if } d_{c-i+1} = d_{c-i+1}^s, \\ 2 & \text{if } d_{c-i+1} > d_{c-i+1}^s. \end{cases}$$

Therefore, four cases may occur that we consider separately by applying Theorem 2.1.

*Case* 1 ($r_1 = r_2 = 2$). One obtains

$$\begin{aligned} t(c-i, d_{c-i}) &= 2d_{c-i} - (c-i+2), \\ t(c-i+1, d_{c-i+1}) &= 2d_{c-i+1} - (c-i+3), \\ t(c-i+1, d_{c-i+1}^n) &= 2(d_{c-i+1} + d_{c-i}) - (c-i+3). \end{aligned}$$

This implies

$$\begin{aligned} &t(c-i+1, d_{c-i+1} + d_{c-i}) - t(c-i+1, d_{c-i+1}) - t(c-i, d_{c-i}) \\ &= 2(d_{c-i+1} + d_{c-i}) - (c-i+3) - 2d_{c-i+1} + (c-i+3) - 2d_{c-i} + (c-i+2) \\ &= 2c - 2i + 5 - (c-i+3) = c - i + 2. \end{aligned}$$

*Case* 2 ($r_1 = 2$ and $r_2 = 1$). It follows that

$$\begin{aligned} t(c-i, d_{c-i}) &= 2d_{c-i} - (c-i+2), \\ t(c-i+1, d_{c-i+1}) &= d_{c-i+1} - 1, \\ t(c-i+1, d_{c-i+1}^n) &= 2(d_{c-i+1} + d_{c-i}) - (c-i+3). \end{aligned}$$

This implies

$$\begin{aligned} &t(c-i+1, d_{c-i+1} + d_{c-i}) - t(c-i+1, d_{c-i+1}) - t(c-i, d_{c-i}) \\ &= 2(d_{c-i+1} + d_{c-i}) - (c-i+3) - d_{c-i+1} + 1 - 2d_{c-i} + (c-i+2) \\ &= d_{c-i+1} = d_{c-i+1}^s = c - i + 2. \end{aligned}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

*Case* 3 ($r_1 = 1$ and $r_2 = 2$). One has

$$\begin{aligned} t(c-i, d_{c-i}) &= d_{c-i} - 1, \\ t(c-i+1, d_{c-i+1}) &= 2d_{c-i+1} - (c-i+3), \\ t(c-i+1, d^n_{c-i+1}) &= 2(d_{c-i+1} + d_{c-i}) - (c-i+3). \end{aligned}$$

This implies

$$\begin{aligned} &t(c-i+1, d_{c-i+1} - d_{c-i}) - t(c-i+1, d_{c-i+1}) - t(c-i, d_{c-i}) \\ &= 2(d_{c-i+1} + d_{c-i}) - (c-i+3) - 2d_{c-i+1} + (c-i+3) - d_{c-i} + 1 \\ &= d_{c-i} + 1 = d^s_{c-i} + 1 = c - i + 2. \end{aligned}$$

*Case* 4 ($r_1 = r_2 = 1$). One has that

$$\begin{aligned} t(c-i, d_{c-i}) &= d_{c-i} - 1, \\ t(c-i+1, d_{c-i+1}) &= d_{c-i+1} - 1, \\ t(c-i+1, d^n_{c-i+1}) &= 2(d_{c-i+1} + d_{c-i}) - (c-i+3). \end{aligned}$$

This implies

$$\begin{aligned} &t(c-i+1, d_{c-i+1} + d_{c-i}) - t(c-i+1, d_{c-i+1}) - t(c-i, d_{c-i}) \\ &= 2(d_{c-i+1} + d_{c-i}) - (c-i+3) - d_{c-i+1} + 1 - d_{c-i} + 1 \\ &= d_{c-i+1} + d_{c-i} + 2 - (c-i+3) = d^s_{c-i+1} + d^s_{c-i} + 2 - (c-i+3) = c - i + 2. \end{aligned}$$

Finally, we consider (A4) and (A5), i.e., $d^n_2 = 3$ and $d^n_1 = 2$. One obtains for $d^n_1$ that $t(1, d^n_1) = 1$ and for $d^n_2$ that $t(2, d^n_2) = 2$.

Due to (2.4), it follows that $\hat{t}(c, l+3) = \sum_{j=1}^{c} t(j, d^n_j) + (l+3)$. We split the indices into $\{j = 1, \dots, c\} = \{c, \dots, c-i+2\} \cup \{c-i+1\} \cup \{c-i, \dots, 3\} \cup \{2, 1\}$. Including the formulas obtained for (A1)–(A5) yields

$$\begin{aligned} \hat{t}(c, l+3) &= \sum_{j=c-i+2}^{c} t(j, d_j) + t(c-i+1, d_{c-i+1}) + t(c-i, d_{c-i}) + (c-i+2) \\ &\quad + \sum_{j=3}^{c-i} (t(j-1, d_{j-1}) - 1) + t(2,3) + t(1,2) + (l+3) \\ &= \sum_{j=c-i}^{c} t(j, d_j) + (c-i+2) + \sum_{j=2}^{c-i-1} (t(j, d_j) - 1) + t(2,3) + t(1,2) + (l+3). \end{aligned}$$

Since $d_1 = 2$, we obtain

$$\begin{aligned} \hat{t}(c, l+3) &= \sum_{j=1}^{c} t(j, d_j) + t(2,3) + (c-i+2) - (c-i-2) + (l+3) \\ &= \sum_{j=1}^{c} t(j, d_j) + l + 9. \end{aligned}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Therefore, we obtain with (2.4) that $\hat{t}(c, l+3) - \hat{t}(c, l) = 9$ holds. Using $t(c, l+3) = t(c, l) + 9$ and that the repetition number $r$ for $l+3$ is $r = 3$, one has

$$\hat{t}(c, l+3) - t(c, l+3) = \hat{t}(c, l) - t(c, l) = 1,$$

proving the assertion. The condition $d_j > d_j^s$ for $j = 2, \ldots, c-i-1$ is absolutely necessary, unless the last two checkpoints are replaced. If $d_j > d_j^s$, one has $r = 1$ for at least one $j$. Then $t(j, d_j^n) = t(j-1, d_{j-1})$ holds, since $t(., d_j)$ is independent of the number of the checkpoints. In this case, no checkpoint distribution can be found with $\hat{t}(c, l+3) = t(c, l+3) + 1$.

Now we consider the second rule. Storing the new state by overwriting the content of checkpoint $c-1$ implies that $d_2^n = 6$ and $d_1^n = 2$. Thus, $t(2, d_2^n) = t(2, 6) = 8$ and $t(1, d_1^n) = t(1, 2) = 0$. Then we have

$$\hat{t}(c, l+3) - \hat{t}(c, l) = t(2, 6) - t(2, 3) + 3 = 8 - 2 + 3 = 9,$$

and hence

$$\hat{t}(c, l+3) - t(c, l+3) = \hat{t}(c, l) - t(c, l) = 1,$$

proving the assertion. □

The resulting online checkpointing strategy extends Algorithm I. First, the checkpoint distribution for $l = u_3$ must be calculated. All the corresponding intermediate states have to be stored during the forward integration and must not be overwritten. The algorithm starts by overwriting the content of the last checkpoint $c-1$ with the state $x_{u_2}$ in the time integration from $x_{u_2}$ to $x_{u_2+1}$ due to Proposition 2.2. Then, one performs three time steps (Proposition 2.1) and chooses a checkpoint that fulfills the condition of Theorem 2.3. We propose to always select the content of the largest checkpoint to be replaced. Since all states of the checkpoint distribution for $l = u_3$ have to be stored in a checkpoint, sometimes only one time step is executed before modifying the checkpoint distributions. In this case two different situations occur for $l+1$.

1. The last replaced checkpoint for $l$ and $l-3$ was the same. Then $d_2 = 6$. In this situation, we replace the content of the last checkpoint again yielding $d_2 = 7$ and $t(2, d_2) = 11$, and the equality $\hat{t}(c, l) = t(c, l) + 1$ holds for $l$. This distribution is optimal for $l+1$ and $l+2$ due to (2.1).
2. The replaced checkpoints for $l$ and $l-3$ were different. Then $d_2 = 3$. In this situation, we replace the content of the last checkpoint. Then $d_2 = 4$, $t(2, d_2) = 4$, and the equality $\hat{t}(c, l) = t(c, l) + 2$ holds. Furthermore, $\hat{t}(c, l) = t(c, l) + 1$ is valid for $l+1$ and $l+2$.

This yields the following online checkpointing algorithm for $r = 3$. Here, the variable $f$ denotes the number of time steps that have to be performed before a new state is replaced in a checkpoint.

**Algorithm II:** Online Checkpointing Algorithm for $r = 3$:

**Start:** Set $f = 0$ and $\mathbf{C} = \{d_i^s | i = 1, \ldots, c\}$ and $\mathbf{D} = \{d_i^e | i = 1, \ldots, c\}$
**for** $l = u_2, \ldots$
1. **If** $l = u_3$, then
   switch to checkpointing algorithm proposed in [18]
2. Evaluate $x_{l+1} = F_l(x_l)$
3. **If** termination criterion fulfilled, **then**
   start reversal

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

4. **If** $f = 0$, **then**
   **If** checkpoint $i$ was modified in the last time step, **then**
      Store $x_l$ in checkpoint $i$
   **else**
      Choose the maximum index $i$ such that
         (a) $c_i$ fulfills either rule 1 or rule 2 of Theorem 2.3
         (b) $c_i \notin \mathbf{D}$
   **If** $l + 1 \in \mathbf{D}$, **then** $f = 0$
   **else** $f = 2$
**else** $f = f - 1$

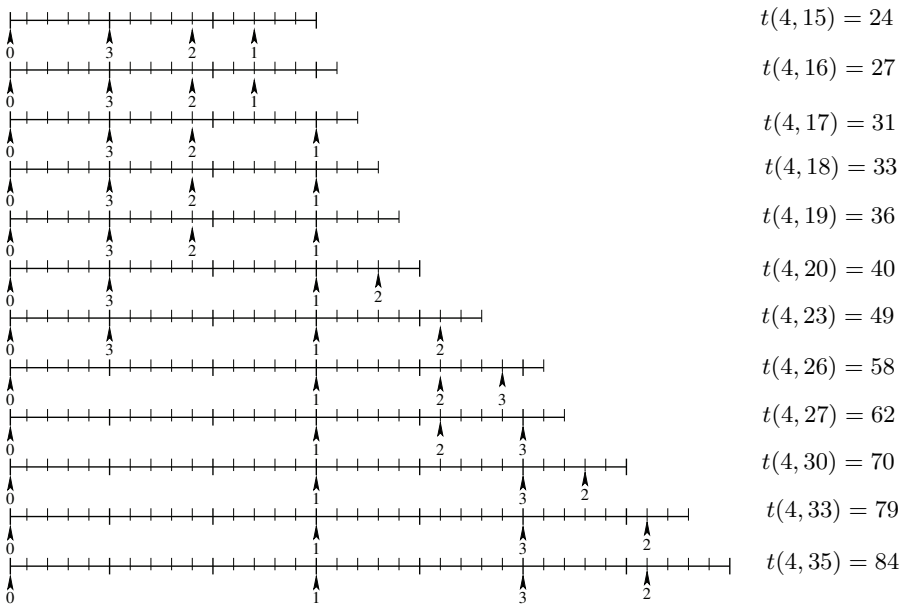



FIG. 2.6. *Checkpoint distributions for* $c = 4$ *and* $l = 15, \ldots, 35$.

Figure 2.6 illustrates the application of Algorithm II as an extension of Algorithm I for $c = 4$ and $15 \leq l \leq 35$ time steps to be reversed. The checkpoint distribution for $l = 15$ is the result of Algorithm I. Due to Proposition 2.2 this distribution remains unchanged for $l = 16$ since it is optimal. If we then discover that the time integration must be continued, we still have $x_{15}$ stored in the main memory. Hence, applying Theorem 2.3 (replacement condition), we can overwrite the content of checkpoint 1 by the state $x_{15}$. This yields that for $l = 17$ this new distribution has a gap of optimality of one due to Theorem 2.3. For the next two time steps we leave this distribution unchanged since it is optimal due to Proposition 2.1. In the next time step we choose checkpoint two to be overwritten. This new distribution has a gap to optimality of one (Theorem 2.3). Again, we leave this distribution unchanged for the next two time steps, and so on. One exception takes place for $l = 26$. Since 25 belongs to the optimal distribution of $\mathbf{D}$, we have to store $x_{25}$. Since for $l = 25$ and for $l = 22$ the replaced checkpoints were 3 and 2, respectively, we face situation 2. Therefore, we overwrite the contents of checkpoint 3 by $x_{25}$. Then the algorithm proceeds, and we finally reach the optimal distribution $\mathbf{D}$ for $l = 35$.

If $l$ exceeds $u_3$, one may use the minimal repetition dynamic checkpointing algo-

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

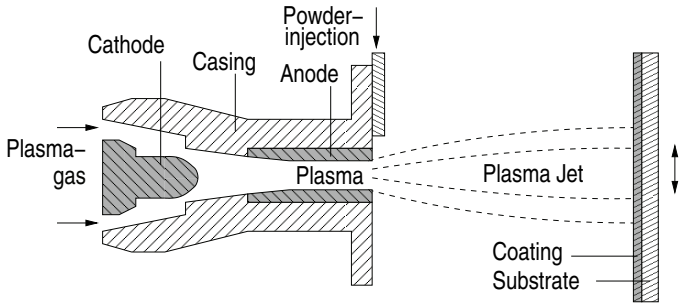



FIG. 3.1. *Plasma spraying procedure.*

rithm presented in [18] as a fall-back option. This dynamic checkpointing algorithm minimizes at least the repetition number $r$ and works for an arbitrary number of time steps. Both algorithms can be combined since the checkpointing distribution for $u_3$ time steps coincides [18].

**3. Numerical results.** This numerical example provided by the Institute for Aerospace Engineering, Technical University Dresden, is motivated by a plasma spraying procedure, as presented in Figure 3.1. Plasma gas is injected into a casing, heated inside, and then pressed out. While the gas is pushed out, powder is injected into the plasma gas. This gas then hits a coating on a substrate. A magnet is installed at the casing, influencing the heat and the output speed of the plasma gas. The aim is to find an optimal heat distribution inside the casing. More details on the problem and the determination of adjoints can be found in [15]. The flow process is described by the compressible Navier–Stokes equations given in matrix form by

$$\partial_t U + \nabla \cdot \vec{F} = \nabla \cdot \vec{D} + S(q) \tag{3.1}$$

with the preservation variable $U = (\rho, \rho v_x, \rho v_y, \rho e)$, the advective fluxes $F$, the diffusive fluxes $D$, and the source term $S$ that depends on a control $q$. This yields the quasi-linear formulation with respect to the primitive variables $\Pi = (v, T)$. Discretizing (3.1) by a discontinuous Galerkin method with symmetric interior penalization the primal formulation reads

$$\begin{aligned}\int_\Omega \varphi \partial_t U \mathrm{d}\Omega = &\int_\Omega \nabla\varphi \cdot (\vec{F} - \vec{D}(U, \nabla\Pi))\mathrm{d}\Omega - \int_\Gamma \varphi(\hat{F}_n - \hat{D}_n)\mathrm{d}\Gamma \\ &- \frac{1}{2}\int_\Gamma \nabla\varphi \cdot \vec{C}_n(\Pi^+ - \Pi^-)\mathrm{d}\Gamma + \int_\Omega \varphi S \mathrm{d}\Omega,\end{aligned}$$

where $F_n$ describes the advective numerical flux and $D_n$ the diffusive numerical flux. Using the spectral element method, the evaluation of the integrals is performed by the Gauss–Legendre–Lobatto quadrature rule on a reference finite element. The time integration is performed by a third order explicit TVD Runge–Kutta method [3] with an adaptive time-stepping scheme [7].

As a very simple model problem of the plasma spraying procedure, we consider the domain shown in Figure 3.2.

We assume there is a steady flow between two isothermal plates positioned at $y = \pm a$ and the flow is driven by the constant body force $\vec{f} = f\vec{e}_x$. The problem is solved by considering the full equations (3.1) in the domain $\Omega = (0, a) \times (-a, a)$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

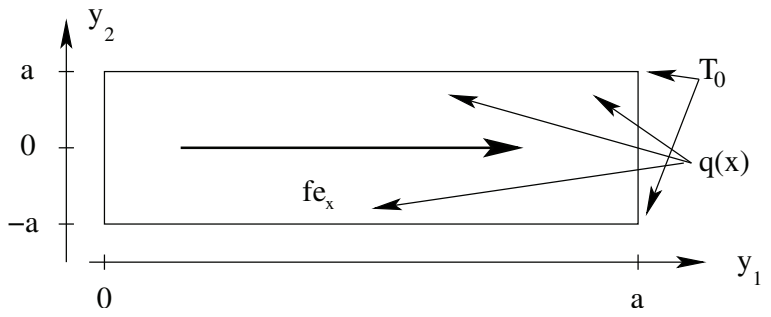



FIG. 3.2. *The flow control problem.*

with boundary conditions $\vec{v}(\pm a) = 0$ and $T(\pm a) = T_0$ at the walls. A related control problem is the determination of a heat source distribution $q(y)$ such that $T[q] = T_0$, i.e.,

$$f(q) = \int_0^{t_F} \int_\Omega (T[q] - T_0)^2 \mathrm{d}\Omega \; \mathrm{d}t \longrightarrow 0.$$

For our numerical example, we set $a = 0.001$, $f = 4000$, $T_0 = 100$, and $t_F = 5 * 10^{-5}$. Then, the final time $t_F$ is reached after 853 time steps. Let $l$ be the number of time steps to be performed and $x_i$ the states at time $t_i = t_0 + ih$ with $x_i = (y_1, y_2)$. Using the basic approach of storing all intermediate states, one obtains the following time stepping procedure:

**for** $i = 0, 1, \ldots$
  **forward_timestep(x)**
  **test stopping criterion**

**3.1. Software.** Using the basic approach of storing all intermediate states, one has the following procedure to compute the desired adjoint values. Let $bx_0, \ldots, bx_{l-1}$ be the corresponding adjoint states and $bq_0, \ldots, bq_{l-1}$ the adjoint controls. Then the time-stepping procedure with storing every state reads

**for** $i = 0, 1, \ldots$
  **store (x,i)**
  **forward_timestep(x)**
  **test stopping criterion**
**init(x,bx,bq)**
**for** $i = l - 1, 0, -1$
  **restore(x,i)**
  **adjoint_timestep(x,bx,bq)**

The adjoints for one time step are determined by automatic differentiation (AD) [5]. To generate the routine **adjoint_timestep**, we apply the AD-enabled NAGWare Fortran compiler developed by the University of Hertfordshire and RWTH Aachen University in collaboration with NAG Ltd. The compiler provides forward [14] and reverse modes [13] of AD by operator overloading as well as by source transformation [12].

The software **revolve** [4] has been extended to also provide the online checkpointing approaches presented here. It can be downloaded from http://www2.math.uni-paderborn.de/index.php?id=12067&L=1. In this software, the optimal online checkpointing of Algorithm I, the almost optimal online checkpointing of Algorithm II, and

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

the heuristic approach of **arevolve** [10] are combined so that one is able to integrate this checkpointing software into an adaptive time-stepping procedure. Once the reversal, i.e., the first adjoint step, is initiated, the optimal reversal strategy already implemented by **revolve** is used for steering the adjoint calculation.

**3.2. Runtime behavior.** For our runtime studies we used a Linux NetworX system with two AMD Opteron Processors for each node and a 4GB main memory. The code runs only on one processor. We use this cluster because the memory access is very fast. The memory requirement for one intermediate state of the forward trajectory, i.e., one checkpoint, is about 10MBs. This yields that the memory requirement for storing the complete forward trajectory is more than 8GBs. Thus, the entire forward solution does not fit into the main memory. We study the effects of online checkpointing by comparing the normalized runtimes, i.e., $\mathrm{TIME}(\nabla f)/\mathrm{TIME}(f)$, for one gradient evaluation by the online checkpointing presented here versus optimal offline checkpointing (**revolve**) and versus the heuristic online checkpointing (**arevolve**) under the assumption that the forward integration is performed by an adaptive time stepping. To use **revolve** we perform one pure forward integration to determine the required number of forward time steps. However, we do not include this extra forward integration in the time measurements for the adjoint computation. This makes the comparison of online checkpointing versus offline checkpointing possible so that one can see the effects that are only because of the different checkpointing procedures. Our primary goal is to achieve the same runtime for our online checkpointing algorithm compared to the runtime of **revolve**.

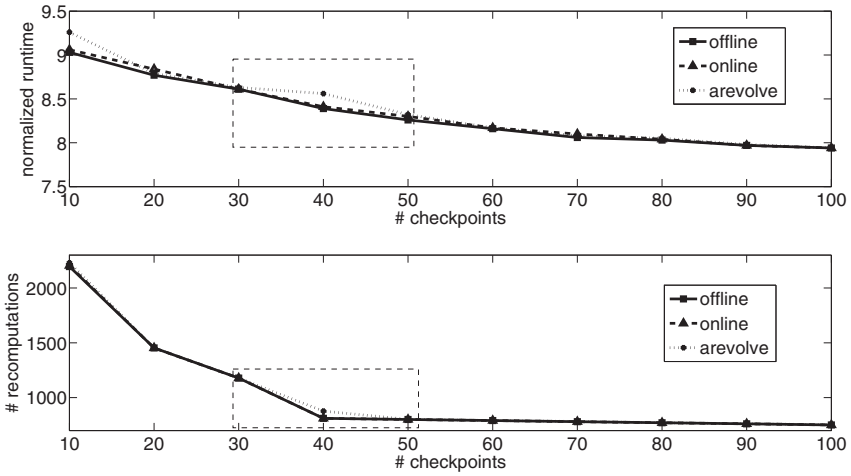



FIG. 3.3. *Comparison of normalized runtime and number of recomputations required by optimal offline checkpointing, (almost) optimal online checkpointing, and heuristic online checkpointing.*

Figure 3.3 illustrates the runtime behavior and the number of recomputations of the three checkpointing approaches for 853 time steps to perform and up to 100 checkpoints yielding a repetition rate of $r \leq 3$. Hence, Algorithm I is used for from 40 up to 100 checkpoints and Algorithm II for from 16 up to 40 checkpoints. We observe typical behavior of checkpointing approaches; i.e., the runtime and the number of recomputations decrease when the number of checkpoints increases. One can see that the runtimes of optimal offline checkpointing and the two online checkpointing

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

approaches almost coincide for a wide range. The runtime of the (almost) optimal online checkpointing procedures is only a little bit higher, which may result from a different number of write counts, i.e., the number of times an intermediate state is stored in a checkpoint during the forward integration. The ratio of write counts of online versus offline checkpointing during the forward integration is illustrated in Figure 3.4. This may be one reason for the higher runtime needed by the (almost) optimal online checkpointing.

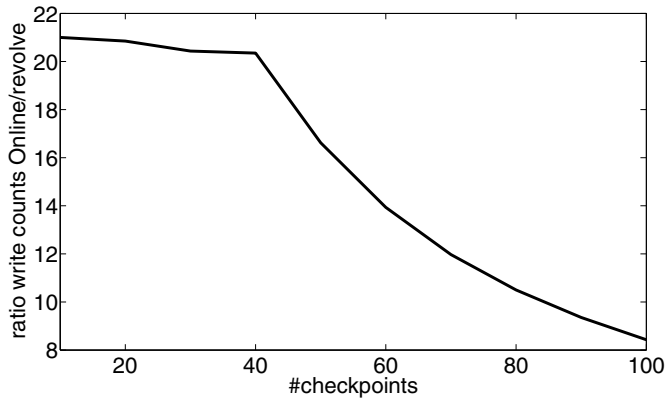



FIG. 3.4. *Ratio of write counts during the forward integration, i.e., the number of write counts (online checkpointing) versus the number of write counts (offline checkpointing).*

The ratio of write counts of the (almost) optimal online versus the offline checkpointing during the forward integration almost stays constant for up to 40 checkpoints, and then it decreases. This is due to the fact that if more than 40 checkpoints are used, the repetition rate is 2. In the other cases, i.e., for less than 40 checkpoints, $r = 3$. Hence, the number of stored checkpoints during the forward integration performed by the (almost) optimal online checkpointing proposed here is about 20 times higher than that during the performance of optimal offline checkpointing. In another paper the authors analyzed the determination of write and read counts for optimal offline checkpointing distributions [17]. This forms the basis for the so-called multistage checkpointing approach that can be extended to a general multistage online checkpointing approach.

Another result of Figure 3.3 is the runtime behavior of **arevolve** versus (almost) optimal online checkpointing and optimal offline checkpointing. We observe that the runtimes coincide in wide ranges. Around 40 checkpoints, one finds a remarkable increase of runtime and the number of recomputations needed by **arevolve** compared to **revolve** and the online checkpointing proposed here. A corresponding zoom can be seen in Figure 3.5.

It follows that for 32 checkpoints the runtimes almost coincide, and then for up to 42 checkpoints the difference in runtime increases. After 42 checkpoints this difference is almost zero again. This higher difference stems from a higher number of extra forward time steps required by **arevolve** compared to the minimal number computed by **revolve** as illustrated in Figure 3.6. One can see that in the range from 34 to 42 checkpoints this difference is high. As soon as it drops almost to zero, i.e., when **arevolve** needs almost the minimal number of extra forward steps, the runtime behavior of both approaches is nearly identical.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

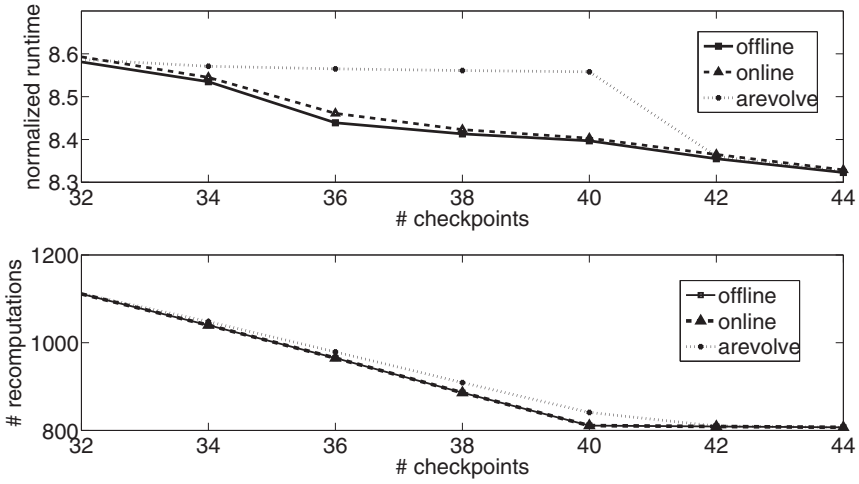



FIG. 3.5. *Comparison of normalized runtime and number of recomputations required by optimal offline checkpointing, (almost) optimal online checkpointing, and heuristic online checkpointing for* 32–44 *checkpoints.*

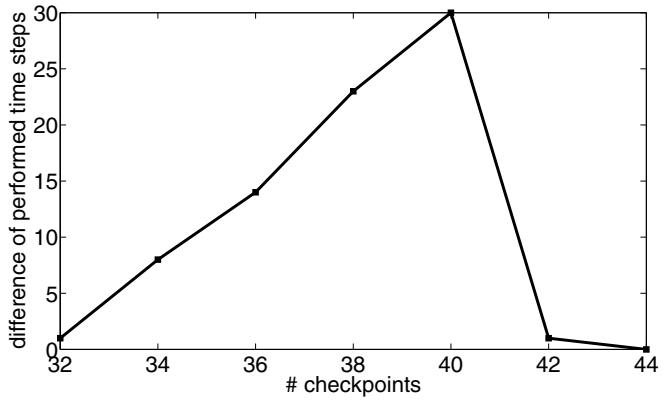



FIG. 3.6. *Comparison of extra time steps performed by heuristic online checkpointing and optimal offline checkpointing.*

**4. Conclusion and outlook.** In this paper we derived two algorithms. We have shown that if the repetition rate is 2, one can apply optimal online checkpointing for $0 \leq l \leq \beta(c, 2)$, yielding a minimal number of extra forward time steps. In addition, we have shown that if the repetition number is 3, one cannot compute optimal checkpoint distributions over $\beta(c, 2) < l \leq \beta(c, 3)$, but it is possible to create distributions that are optimal for a large range of $l$. If they are not optimal, the difference to a minimal runtime for the adjoint computation is no more than the execution of two time steps. The application of the algorithm for $r = 3$ needs the user to keep the state $x_l$ for the integration step $x_l \to x_{l+1}$ in memory; hence the old and new states must be available. In our numerical example we examined the runtime behavior of different online and optimal offline checkpointing distributions. It was shown that the runtimes almost coincide with only a slight increase of runtime by the online checkpointing

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

approach. Then we compared the observed runtime behavior to the runtimes needed by **arevolve** and observed that there is also a slight increase of runtime compared to **revolve** in wide range. However, as was also shown, the runtime can be higher in ranges where the number of forward time steps needed by **arevolve** is much higher than the minimal number. Our next step in future work is to combine the multistage checkpointing approach proposed in [17] and the results of the present paper. This will result in a so-called multistage online checkpointing.

**Acknowledgment.** We thank the anonymous referees for their valuable remarks and suggestions.

## REFERENCES

[1] R. Becker, D. Meidner, and B. Vexler, *Efficient numerical solution of parabolic optimization problems by finite element methods*, Optim. Methods Softw., 22 (2007), pp. 813–833.

[2] R. Becker and R. Rannacher, *An optimal control approach to a posteriori error estimation in finite element methods*, Acta Numer., 10 (2001), pp. 1–102.

[3] B. Cockburn, C. Johnson, C.-W. Chu, and E. Tedmor, *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, Springer, New York, 1998.

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

[5] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, 2nd ed., SIAM, Philadelphia, 2008.

[6] M. D. Gunzburger, *Perspectives in Flow Control and Optimization*, Adv. Des. Control 5, SIAM, Philadelphia, 2002.

[7] E. Hairer, S. P. Norsett, and G. Wanner, *Solving Ordinary Differential Equations* I, Springer, New York, 1993.

[8] V. Heuveline and A. Walther, *Online checkpointing for parallel adjoint computation in PDEs: Application to goal-oriented adaptivity and flow control*, in Euro-Par 2006 Parallel Processing, W. E. Nagel, W. V. Walter, and W. Lehner, eds., Springer, New York, 2006, pp. 689–699.

[9] M. Hinze and K. Kunisch, *Second order methods for optimal control of time-dependent fluid flow*, SIAM J. Control Optim., 40 (2001), pp. 925–946.

[10] M. Hinze and J. Sternberg, *A-revolve: An adaptive memory and run-time-reduced procedure for calculating adjoints; with an application to the instationary Navier-Stokes system*, Optim. Methods Softw., 20 (2005), pp. 645–663.

[11] K. Kubota, *A Fortran 77 preprocessor for reverse mode automatic differentiation with recursive checkpointing*, Optim. Methods Softw., 10 (1998), pp. 319–335.

[12] M. Maier and U. Naumann, *Intraprocedural adjoint code generated by the differentiation-enabled NAGWare Fortran compiler*, in Proceedings of the 5th International Conference on Engineering Computational Technology (ECT 2006), Civil-Comp Press, Stirling, UK, 2006, pp. 1–19.

[13] U. Naumann and J. Riehme, *Computing adjoints with the NAGWare Fortran* 95 *compiler*, in Automatic Differentiation: Applications, Theory, and Tools, H. Bücker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, eds., Lect. Notes Comput. Sci. Eng. 50, Springer, Berlin, 2005, pp. 159–170.

[14] U. Naumann and J. Riehme, *A differentiation-enabled Fortran* 95 *compiler*, ACM Trans. Math. Software, 31 (2005), pp. 458–474.

[15] U. Naumann, J. Riehme, J. Stiller, and A. Walther, *Adjoints for time-dependent optimal control*, in Advances in Automatic Differentiation, C. Bischof, M. Bücker, P. Hovland, U. Naumann, and J. Utke, eds., Lect. Notes Comput. Sci. Eng. 64, Berlin, Springer, 2008, pp. 175–185.

[16] R. Serban and A. C. Hindmarsh, *CVODES: An ODE Solver with Sensitivity Analysis Capabilities*, Tech. report UCRL-JP-20039, Lawrence Livermore National Laboratory, Livermore, CA, 2003.

[17] P. Stumm and A. Walther, *Multistage approaches for optimal offline checkpointing*, SIAM J. Sci. Comput., 31 (2009), pp. 1946–1967.

[18] Q. Wang, P. Moin, and G. Iaccarino, *Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation*, SIAM J. Sci. Comput., 31 (2009), pp. 2549–2567.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.