ADJOINT CALCULATION USING TIME-MINIMAL PROGRAM REVERSALS FOR MULTI-PROCESSOR MACHINES

Andrea Walther
Institute of Scientific Computing
Technical University Dresden
awalther@math.tu-dresden.de

Uwe Lehmann
Center for High Performance Computing
Technical University Dresden
lehmann@zhr.tu-dresden.de

Abstract For computational purposes such as debugging, derivative computations using the reverse mode of automatic differentiation, or optimal control by Newton's method, one may need to reverse the execution of a program. The simplest option is to record a complete execution log and then to read it backwards. As a result, massive amounts of storage are normally required. This paper proposes a new approach to reversing program executions. The presented technique runs the forward simulation and the reversal process at the same speed. For that purpose, one only employs a fixed and usually small amount of memory pads called checkpoints to store intermediate states and a certain number of processors. The execution log is generated piecewise by restarting the evaluation repeatedly and concurrently from suitably placed checkpoints. The paper illustrates the principle structure of time-minimal parallel reversal schedules and quotes the required resources. Furthermore, some specific aspects of adjoint calculations are discussed. Initial results for the steering of a Formula 1 car are shown.

Keywords: Adjoint calculation, Checkpointing, Parallel computing

1. Introduction and Notation

For many industrial applications, rather complex interactions between various components have been successfully simulated with computer

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35699-0_19
E. W. Sachs et al. (eds.), *System Modeling and Optimization XX*
© IFIP International Federation for Information Processing 2003

models. This is true for several production processes, e.g. steel manufacturing with regards to various product properties, for example stress distribution. However, the simulation stage can frequently not be followed by an optimization stage, which would be very desirable. This situation is very often caused by the lack or inaccuracy of derivatives, which are needed in optimization algorithms. Hence, enabling the transition from simulation to optimization represents a challenging research task.

The technique of algorithmic or automatic differentiation (AD), which is not yet well enough known, offers an opportunity to provide the required derivative information [5]. Therefore, AD can contribute to overcoming the step from pure simulation and hence "trial and error"-improvements to an exact analysis and systematic derivative-based optimization.

The key idea of algorithmic differentiation is the systematic application of the chain rule. The mathematical specification of many applications involves nonlinear vector functions

$$F : \mathbb{R}^n \to \mathbb{R}^m, \qquad x \mapsto F(x),$$

that are typically defined and evaluated by computer programs. This computation can be decomposed into a (normally large) number of very simple operations, e.g. additions, multiplications, and trigonometric or exponential function evaluations. The derivatives of these elementary operations can be easily calculated with respect to their arguments. A systematic application of the chain rule yields the derivatives of a hierarchy of intermediate values. Depending on the starting point of this methodology, either at the beginning or at the end of the sequence of operations considered, one distinguishes between the forward mode and the reverse mode of AD. The reverse mode of algorithmic differentiation is a discrete analog of the adjoint method known from the calculus of variations.

The gradient of a scalar-valued function is yielded by the reverse mode in its basic form for no more than five times the operations count of evaluating the function itself. This bound is completely independent of the number of independent variables. More generally, this mode allows the computation of Jacobians for at most five times the number of dependents times the effort of evaluating the underlying vector function. However, the spatial complexity of the basic reverse mode, i.e. its memory requirement, is proportional to the temporal complexity of the evaluation of the function itself. This behaviour is caused by the fact that one has to record a complete *execution log* onto a data structure called *tape* and subsequently read this tape backward. For each arith-

metic operation, the execution log contains a code and the addresses of the arguments as well as the computed value. It follows that the practical exploitation of the advantageous temporal complexity bound for the reverse mode is severely limited by the amount of memory required.

The reversal of a given function F is already being extensively used to calculate hand-coded adjoints. In particular, there are several contributions on weather data assimilation (e.g. [11]). Here, the desired gradients can be obtained with a low temporal complexity by integrating the linear co-state equation backwards along the trajectory of the original simulation. This well-known technique is closely related to the reverse mode of AD [3]. Moreover, debugging and interactive control may require the reconstruction of previous states by some form of running the program backwards that evaluates F. The need for some kind of logging arises whenever the process described by F is not invertible or ill conditioned. In these cases one cannot simply apply an inverse process to evaluate the inverse mapping F^{-1}. Consequently, the reversal of a program execution within a reasonable memory requirement has received some (but only perfunctory) attention in the computer science literature (see e.g. [12]).

This paper presents a new approach to reversing the calculation of F. For that reason, in the remainder of this section, the structure of the function F is described in detail. The reversal technique proposed in this article only employs a fixed and usually small amount of memory pads to store intermediate states and a certain number of processors for reversing F in minimal time. The corresponding time-minimal parallel reversal schedules are introduced in Section 2. The simulation of a Formula 1 car is considered in Section 3. The underlying ODE system is introduced. Then two different ways to calculate adjoints are discussed. Subsequently, the initial numerical results are presented. Finally, some conclusions are drawn in Section 4.

Throughout it is assumed that the evaluation of F comprises the evaluation of subfunctions F_i, $1 \leq i \leq l$, called physical steps that act on state x^{i-1} to calculate the subsequent intermediate state x^i for $1 \leq i \leq l$ depending on a control u^{i-1}. Hence, one has

$$x^i \;=\; F_i(x^{i-1}, u^{i-1}) \; .$$

Therefore, F can be thought of as a discrete *evolution*. The intermediate states of the evolution F represented by the counter i should be thought of as vectors of large dimensions. The physical steps F_i describe mathematical mappings that in general cannot be reversed at a reasonable cost even for given u^{i-1}. Hence, it is impossible to simply apply the inverses F_i^{-1} in order to run the program backwards from state l to state 0. It

will also be assumed that due to their size, only a limited number of intermediate states can be kept in memory.

Furthermore, it is supposed that for each $i \in \{1, \ldots, l\}$, there exist functions $\hat{F}_i$ that cause the recording of intermediate values generated during the evaluation of F_i onto the tape and corresponding functions $\bar{F}_i$ that perform the reversal of the ith physical step using this tape. More precisely, one has the reverse steps

$$(\bar{x}^{i-1}, \bar{u}^{i-1}) \;=\; \bar{x}^i \, F_i'(x^{i-1}, u^{i-1}) \;\equiv\; \bar{F}_i(x^{i-1}, u^{i-1}, \bar{x}^i)\,,$$

where F_i' denotes the Jacobian of F_i with respect to x^{i-1} and u^{i-1}. The calculation of adjoints using the basic approach is depicted in Figure 1. Applying a checkpointing technique, the execution log is gen-

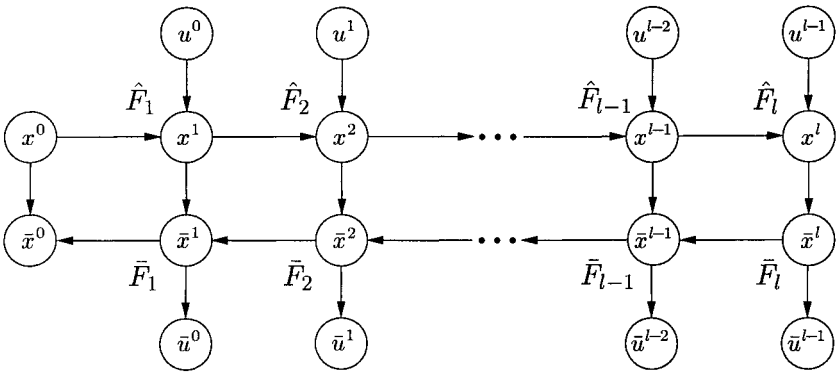


Figure 1. Naïve approach to calculate Adjoints

erated piecewise by restarting the evaluation repeatedly from suitably placed checkpoints, according to requests by the reversal process. Here, the checkpoints can be thought of as pointers to nodes representing intermediate states i. Using a checkpointing strategy on a uni-processor machine, the calculation of F can be reversed, even in such cases where the basic reverse mode fails due to excessive memory requirement (see e.g. [7, 6]). However, the runtime for the reversal process increases compared to the naïve approach. For multi-processor machines, this paper presents a checkpointing technique with concurrent recalculations that reverses the program execution in minimal wall-clock time.

2. Time-minimal Parallel Reversal Schedules

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

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

2.) the number p of processors that are available;

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

4.) the step costs: $\tau = TIME(F_i)$, $\hat{\tau} = TIME(\hat{F}_i)$, $\bar{\tau} = TIME(\bar{F}_i)$.

Well known reversal schedules for serial machines, i.e. $p = 1$, and constant step costs τ allow an enormous reduction of the memory required to reverse a given evolution F in comparison with the basic approach (see e.g. [7, 6]). Even if the step costs $\tau_i = TIME(F_i)$ are not constant it is possible to compute optimal serial reversal schedules [13]. However, one has to pay for the improvements in the form of a greater temporal complexity because of repeated forward integrations.

If no increase in the time needed to reverse F is acceptable, the use of a sufficiently large number of additional processors provides the possibility to reverse the evolutionary system F with drastically reduced spatial complexity and still minimal temporal complexity. Corresponding parallel reversal schedules that are optimal for given numbers l of physical steps, $p > 1$ processors, c checkpoints, and constant step costs were presented for the first time in [13]. For that purpose, it is supposed that $\tau = 1$, $\hat{\tau} \geq 1$, and $\bar{\tau} > 1$, with $\hat{\tau}, \bar{\tau} \in \mathbb{N}$. Furthermore, it is always assumed that the memory requirement for storing the intermediate states is the same for all i. Otherwise, it is not clear whether and how parallel reversal schedules can be constructed and optimized. The techniques developed in [13] can certainly not be applied. In practical applications, nonuniform state sizes might arise, for example as result of adaptive grid refinements, or function evaluations that do not conform naturally to our notion of an evolutionary system on a state space of fixed dimension.

Finding a time-minimal parallel reversal schedule can be interpreted as a very special kind of scheduling problem. The general problem class is known to be NP-hard (e.g. [4]). Nevertheless, it is possible to specify suitable time-minimal parallel reversal schedules for a arbitrary number l of physical steps because the reversal of a program execution has a very special structure. For the development of these time-minimal and resource-optimal parallel reversal schedules, first an exhaustive search algorithm was written. The input parameters were the number p of available processors and the number c of available checkpoints with both $\hat{\tau}$ and $\bar{\tau}$ set to 1. The program then computed a schedule that reverses the maximal number of physical steps $l(p, c)$ in minimal time using no more than the available resources p and c for $p + c \leq 10$. Here, minimal time means the wall clock equivalent to the basic approach of recording

all needed intermediate results. Examining the corresponding parallel reversal schedules, one obtained that for $p > c$, only the resource number $\varrho = p + c$ has an influence on $l(p, c) \equiv l_\varrho$. Therefore, the development of time-minimal parallel reversal schedules, that are also resource-optimal, is focused on a given resource number ϱ under the tacit assumption $p > c$. The results obtained for $\varrho \leq 10$ provided sufficient insight to deduce the general structure of time-minimal parallel reversal schedules for arbitrary combinations of $\hat{\tau} \geq 1$, $\bar{\tau} \geq 1$, and $\varrho > 10$. Neglecting communication cost, the following recurrence is established in [13]:

Theorem: *Given the number of available resources $\varrho = p + c$ with $p > c$ and the temporal complexities $\hat{\tau} \in \mathbb{N}$ and $\bar{\tau} \in \mathbb{N}$ of the recording steps $\hat{F}_i$ and the reverse steps $\bar{F}_i$, then the maximal length of an evolution that can be reverted in parallel without interruption is given by*

$$l_\varrho = \begin{cases} \varrho & \text{if } \varrho < 2 + \hat{\tau}/\bar{\tau} \\ l_{\varrho-1} + \bar{\tau}\, l_{\varrho-2} - \hat{\tau} + 1 & \text{else.} \end{cases} \tag{1}$$

In order to prove this result, first an upper bound on the number of physical steps that can be reversed with a given number ϱ of processors and checkpoints was established. Subsequently, corresponding reversal schedules that attain this upper bound were constructed recursively. For this purpose, the resource profiles of the constructed parallel reversal schedules were analyzed in detail. In addition to the recursive construction of the desired time-minimal reversal schedules, the resource profiles yield an upper bound for the number p of processors needed during the reversal process. To be more precise, for reversing l_ϱ physical steps, one needs no more than

$$p_u = \begin{cases} \left\lceil \frac{\varrho+1}{2} \right\rceil & \text{if } \bar{\tau} \geq \hat{\tau} \\ \left\lceil \frac{\varrho+1}{2} \right\rceil + \left\lceil \frac{1}{2} \left\lfloor \frac{\hat{\tau}-1}{\bar{\tau}} \right\rfloor \right\rceil & \text{else} \end{cases}$$

processors [13]. Hence, roughly half of the resources have to be processors. This fact offers the opportunity to assign one checkpoint to each processor.

A time-minimal reversal schedule for $l = 55$ is depicted in Figure 2. Here, vertical bars represent checkpoints and slanted bars represent running processes. The shading indicates the physical steps F_i, the recording steps $\hat{F}_i$ and the reverse steps $\bar{F}_i$ to be performed.

Based on the recurrence (1), it is possible to describe the behaviour of l_ϱ more precisely. For $\hat{\tau} = \bar{\tau} = 1$, one finds that the formula for l_ϱ is equal to the Fibonacci-number $f_{\varrho-1}$. Moreover, for other combinations of $\hat{\tau}, \bar{\tau} \in \mathbb{N}$, the recurrence (1) produces generalized Fibonacci-numbers

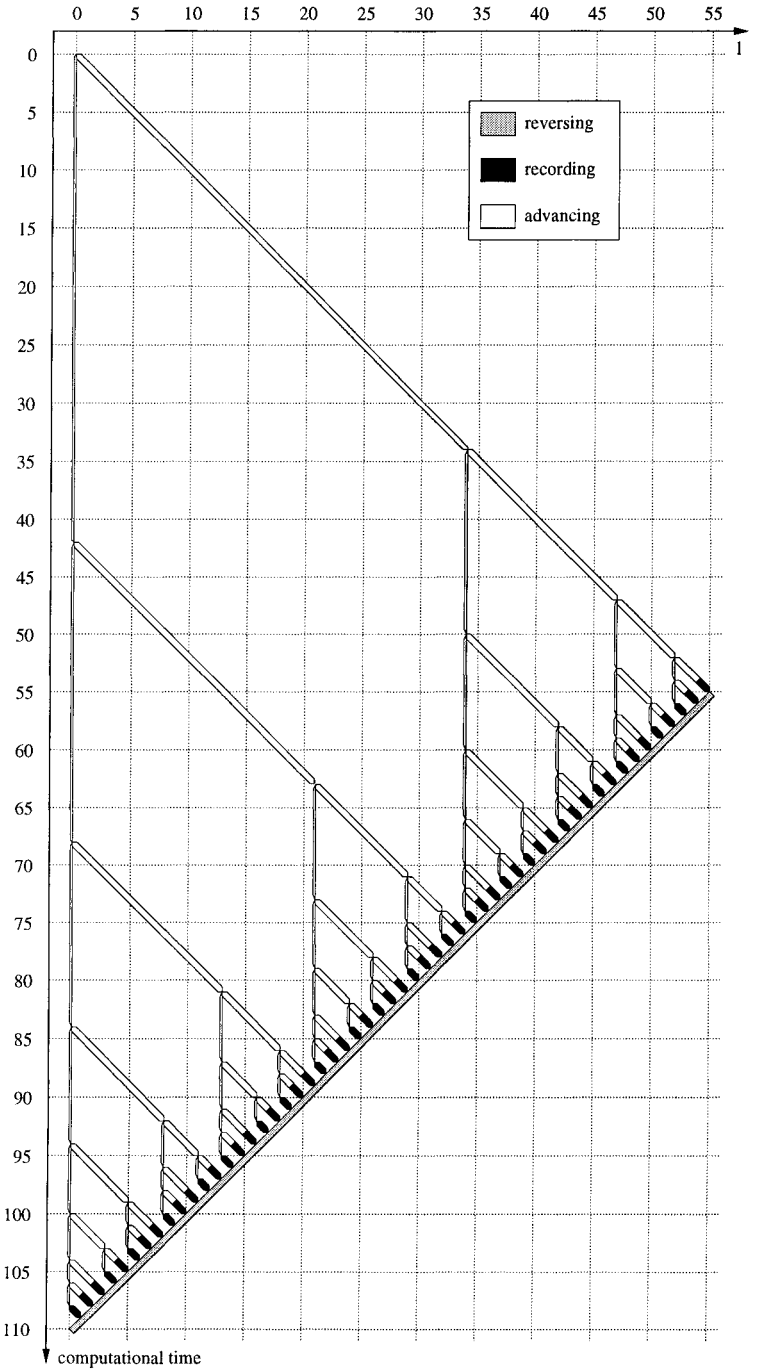


Figure 2. Time-minimal Parallel Reversal Schedule for $l = 21$ and $\hat{\tau} = \bar{\tau} = 1$.

(see e.g. [9]). More specifically, one finds that

$$l_\varrho \sim \frac{1}{2}\left(1+\frac{3}{\sqrt{1+4\bar{\tau}}}\right)\left[\frac{1}{2}(1+\sqrt{1+4\bar{\tau}})\right]^{\varrho-1},$$

in the sense that the ratio between the two sides tends to 1 as ϱ tends to infinity. In the important case $\bar{\tau}=1$ even their absolute difference tends to zero. Thus, $l=l_\varrho$ grows exponentially as a function of $\varrho \approx 2p$ and conversely $p \approx c$ grows logarithmically as a function of l. In order to illustrate the growth of l_ϱ, assume 16 processors and 16 checkpoints are available. These resources suffice to reverse an evolution of $l=2\,178\,309$ physical steps when $\hat{\tau}=\bar{\tau}=1$ and even more steps if $\hat{\tau}=1$ and $\bar{\tau}>1$.

For $\bar{\tau}=1$, i.e., if the forward simulation and the reversal of the time steps can be performed at the same speed, the implementation of this theory was done using the distributed memory programming model [10]. It is therefore possible to run the parallel reversal schedules framework on most parallel computers independent of their actual memory structure. To achieve a flexible implementation, the MPI routines for the communication are used. The parallel reversal schedules are worked off in a process-oriented manner instead of a checkpoint-oriented manner (see [10] for details). This yield the optimal resource requirements of Theorem 1.

In order to apply the parallel reversal schedules framework, one has to provide interfaces and define the main data structures for computing the adjoint. The data structures required are the checkpoints, the traces or tapes, as a result of the recording step $\hat{F}_i$, and the adjoint values. The structure and complexity of this data is independent of the framework since the framework only calls routines such as

- forward(..) for the evaluation of one physical step F_i,
- recording(..) for the evaluation of one recording step $\hat{F}_i$,
- reverse(..) for the evaluation of one reverse step $\bar{F}_i$,

provided by the user. These functions are equivalent to the functions used for a sequential calculation of the adjoint. The index i is an argument of each of the modules. The function recording(..) generates the trace or tape. The function reverse(..) obtains the trace of the last recording step and the adjoint computed so far as arguments. Furthermore, if $i=l$, the function reverse(..) may initialize the adjoints.

Additionally, the user must code communication modules, for example sendCheckpoint(..) and receiveCheckpoint(..). All user-defined routines have to be implemented applying MPI routines. The required process

identifications and message tags are arguments of routines provided by the parallel reversal schedules framework.

3. Model Problem: Steering a Formula 1 Car

In order to test the implementation of parallel reversal schedules, the simulation of an automobile is considered. The aim is to minimize the time needed to travel along a specific road. A simplified model of a Formula 1 racing car [1] is employed. It is given by the ODE system:

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= \frac{(F_{\eta_1}(x,u_2) + F_{\eta_2}(x,u_2))l_f - (F_{\eta_3}(x,u_2) + F_{\eta_4}(x,u_2))l_r}{I} \\
\dot{x}_3 &= \frac{F_{\eta_1}(x,u_2) + F_{\eta_2}(x,u_2) + F_{\eta_3}(x,u_2) + F_{\eta_4}(x,u_2))}{M} - x_2 x_4 \\
\dot{x}_4 &= \frac{F_\xi(x,u_2) - F_a(x)}{M} + x_2 x_3 \\
\dot{x}_5 &= x_4 \sin(x_1) + x_3 \cos(x_1) \\
\dot{x}_6 &= x_4 \cos(x_1) - x_3 \sin(x_1) \\
\dot{x}_7 &= u_1.
\end{aligned}$$

Hence, a go-kart model with rigid suspension and a body rolling about a fixed axis is considered. There are seven state variables representing the yaw angle and rate (x_1, x_2), the lateral and longitudinal velocity (x_3, x_4), global position (x_5, x_6), and the vehicle steer angle (x_7) as shown in Figure 3. The control variables are u_1 denoting the front steer rate and u_2 denoting the longitudinal force as input. The lateral and longitudinal vehicle forces F_η and F_ξ are computed using the state and the control variables as well as the tire forces given by a tire model described in [2]. The force F_a represents the aerodynamic drag depending on the longitudinal velocity. All other values are fixed car parameters such as mass M and length of the car given by l_f and l_r.

In order to judge the quality of the driven line, the cost functional

$$J(s_l) = \int_0^{s_l} S_{cf}(x,s)\big(1 + g(x,s)\big)ds \tag{2}$$

is used. The scaling factor $S_{cf}(x,s)$ changes the original time integration within the cost function to distance integration. Therefore, an integration over the arc length is performed. This variable change has to be done because the end time t_l of the time integration is the value one actually wants to minimise. Hence, t_l is unknown. The computation of the scaling factor $S_{cf}(x,s)$ is described in [1]. The function $g(x,s)$ measures whether or not the car is still on the road. The road is defined by

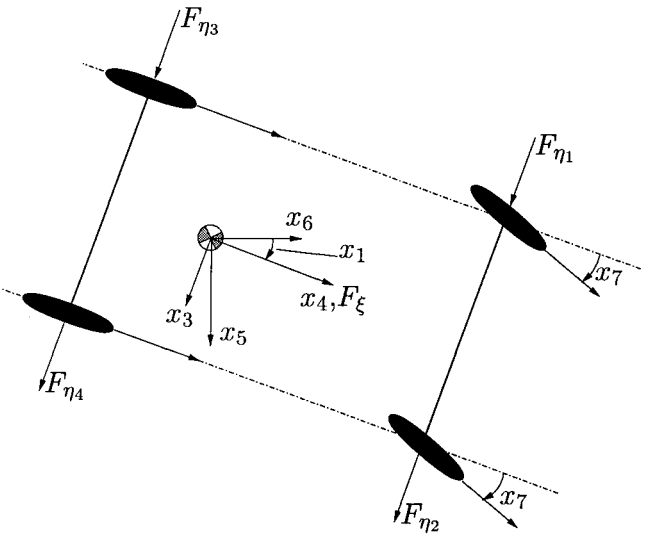


Figure 3. Model of Formula 1 Car.

the road centre line and road width. In the example presented here, the road width is constant a 2.5 m along the whole integration path. The function $g(x, s)$ returns zero as long as the car drives within the road boundaries. If the car leaves the road then $g(x, s)$ returns the distance from the car to the road boundary squared.

3.1. The Forward Integration

For the numerical results presented here, a discretization has to be applied. Therefore, an appropriate initial vector x^0 and the starting position $s^0 = 0$ were chosen. The route is divided equidistantly with a step size of $h = 10\ cm$. The well known four-stage Runge-Kutta scheme

$$\begin{aligned} k_1 &= f(x^{i-1}, u(s^{i-1})) \\ k_2 &= f(x^{i-1} + hk_1/2, u(s^{i-1} + h/2)) \\ k_3 &= f(x^{i-1} + hk_2/2, u(s^{i-1} + h/2)) \\ k_4 &= f(x^{i-1} + hk_3, u(s^{i-1} + h)) \\ x^i &= x^{i-1} + h(k_1 + 2k_2 + 2k_3 + k_4)/6 \end{aligned} \tag{3}$$

serves as physical step F_i for $i = 1, \ldots, 1000$.

The calculations of a physical step F_i form the forward(..)-routine needed by the time-minimal parallel reversal schedules. As mentioned above, in addition to this, one has to provide two further routines,

namely recording(..) and reverse(..). The content of these two modules is described in the next subsection.

3.2. Calculating Adjoints

There are two basic alternatives for calculating the adjoints of a given model. Firstly, one may form the adjoint of the continuous model equation and discretize the continuous adjoint equation. Secondly, one may use automatic differentiation (AD), or hand-coding, to adjoin the discrete evaluation procedure of the model. Both ways do not commute in general (see e.g. [8]). Therefore, one has to be careful when deciding how to calculate the desired adjoints. For the computations shown below, the second option was applied, namely the adjoining of the discretized equation (3). Application of AD in reverse mode amounts to the following adjoint calculation $\bar{F}_i$ (see e.g. [5]):

$$
\begin{aligned}
\tilde{k}_j &= \frac{\partial}{\partial x} k_j & \hat{k}_j &= \frac{\partial}{\partial u} k_j & 1 \le j \le 4 \\
a_4 &= h\bar{x}^i/6 & b_4 &= a_4\tilde{k}_4 & \\
a_3 &= h\bar{x}^i/3 + hb_4 & b_3 &= a_3\tilde{k}_3 & \\
a_2 &= h\bar{x}^i/3 + hb_3/2 & b_2 &= a_2\tilde{k}_2 & \\
a_1 &= h\bar{x}^i/3 + hb_2/2 & b_1 &= a_1\tilde{k}_1 & \\
\bar{u}^i &= \bar{u}^i + a_4\hat{k}_4 & \bar{u}^{i-1} &= a_1\hat{k}_1 & \\
\bar{x}^{i-1} &= \frac{\partial J}{\partial x^{i-1}} + \bar{x}^i + b_1 + b_2 + b_3 + b_4, & & &
\end{aligned}
\tag{4}
$$

for $i = l, \ldots, 1$, where the functions k_j, $1 \le j \le 4$, are defined as in (3). Here, $\bar{u}^i$ denotes the adjoint of the control u at s^i. Note that the integration of the adjoint scheme (4) has to be performed in reverse order starting at $i = l$. One uses $\bar{x}_i^l = \partial J/\partial x^l$, $1 \le i \le 7$ and $\bar{u}_i^l = 0$, $i = 1, 2$ as initial values because of the influence on the cost functional (2). After the complete adjoint calculation, each value $\bar{u}^i$ denotes the sensitivity of the cost functional J with respect to the value u_i.

Now the return value of the routine reverse(..) is clear. It has to contain the computations needed to perform an adjoint step $\bar{F}_i$ according to (4). However, there are two ways to implement the interface between the modules recording(..) and reverse(..). One can either store the stages k_j, $1 \le j \le 4$, during the evaluation of the recording step $\hat{F}_i$. Then the corresponding reverse step $\bar{F}_i$ comprises all calculations shown in (4), i.e. also the computation of the Jacobians $\tilde{k}_j$, $1 \le j \le 4$. As an alternative, one can compute the Jacobians $\tilde{k}_j$, $1 \le j \le 4$ in the recording step $\hat{F}_i$ and store this information on the tape. Then the appropriate reverse

step $\bar{F}_i$ only has to evaluate the last three statements of Equation (4). The runtimes represented here are based on the second approach in order to achieve $\bar{\tau} = 1$. As a result, $\hat{\tau}$ equals 5. This implementation has the advantage that the value of $\bar{\tau}$ and hence the wall clock time are reduced at the expense of $\hat{\tau}$. This can be seen for example in Figure 2, where an increase of $\bar{\tau}$ would result in an bigger slope of the bar describing the adjoint or reverse computations.

As mentioned above, one has to be careful about the adjoint calculation because of the lack of commutativity between adjoining and discretizing in general. Therefore, it is important to note that the Runge-Kutta scheme (3) belongs to a class of discretizations, for which both possibilities of adjoint calculation coincide, giving the same result [8].

3.3. Numerical Results

To test the parallel reversal schedule framework, one forward integration of the car model shown in Figure 4 and one adjoint calculation were performed. As previously mentioned, the integration distance was 100 m and the step size 10 cm. Hence, there are 1000 forward steps F_i. The Figure 5(a) shows the growth of the cost functional for which

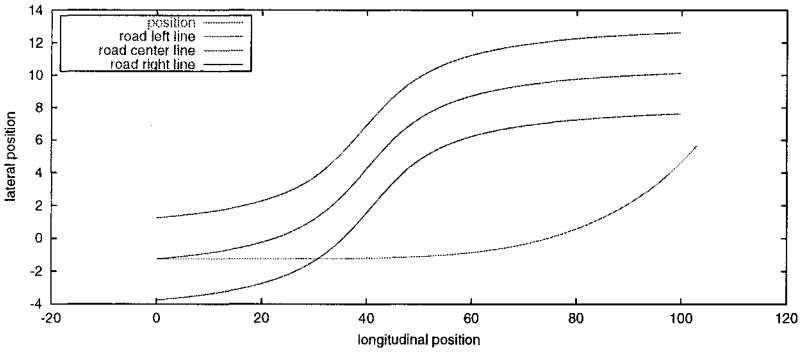
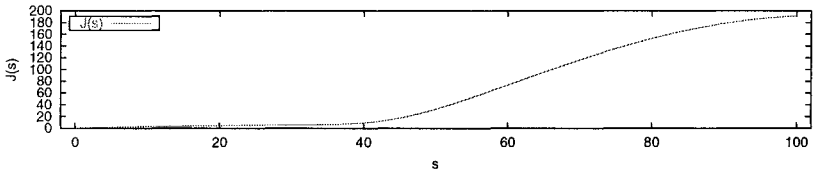
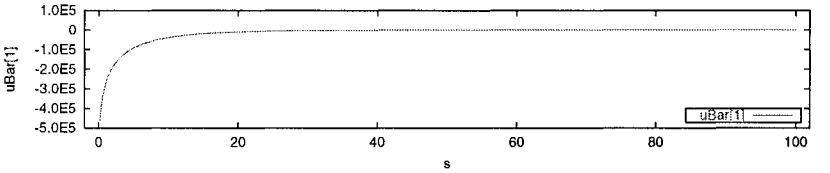


Figure 4. Position of Formula 1 Car.

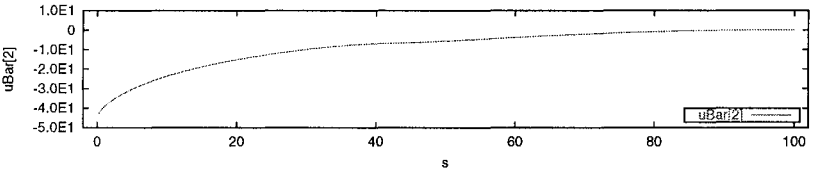
we computed the sensitivities of the control variables u_1 (Figure 5(b)) and u_1 (Figure 5(c)). However, the resource requirements are of primary interest. One integration step in the example is relatively small in terms of computing time. In order to achieve reasonable timings 18 integration steps form one physical step of the parallel reversal schedule. The remaining 10 integration steps were spread uniformly. Hence, one obtains 55 physical steps. Therefore, five processors were needed for the corresponding time-minimal parallel reversal schedule for $\hat{\tau} = \bar{\tau} = 1$. This reversal schedule is with small modifications also nearly optimal for the



(a) Cost Functional $J(s)$.



(b) Adjoint of steering rate u_1.



(c) Adjoint of longitudinal force u_2.

Figure 5. Cost Functional and Adjoint of Control Variables.

considered combination $\hat{\tau} = 5$ and $\bar{\tau} = 1$. A sixth processor (master) was used to organise the program run.

		naïve approach	parallel checkpointing
double variables needed		266010	5092
memory required	in kByte	2128.1	40.7
	in %	100.0	1.9

Table 1. Memory Requirement

The main advantage of the parallel reversal schedules is the enormous reduction in memory requirement as illustrate in Table 1. It shows that for this example, less than a fiftieth of the original memory requirement

is needed, i.e., less than 2%. On the other hand, only six times the original computing power, i.e., processors, is used.

The theoretical runtime is also confirmed by the example as can be seen in Table 2. Due to the slower memory interface on a Cray T3E, the usage of less memory in parallel causes an enormous decrease in runtime. On the other hand the problem is too small and the SGI Origin 3800 too fast to show this effect. Nevertheless, one obtains that the assumption of negligible communication cost is reasonable. This is caused by the fact that the processors have the duration of one full physical step to send and receive a checkpoint because the checkpoint is not needed earlier. Only if the send and receive of one checkpoint needs more time than one physical step the communication cost becomes critical.

		naïve approach	parallel checkpointing
T3E	in sec.	20.27	18.91
	in %	100.0	93.3
Origin 3800	in sec.	6.71	6.04
	in %	100.0	90.0

Table 2. Runtime results

4. Conclusions

The potentially enormous memory requirement of program reversal by complete logging often causes problems despite the ever increasing size of memory systems. This paper proposes an alternative method, where the memory requirement can be drastically reduced by keeping at most c intermediate states as checkpoints. In order to avoid an increase in runtime, p processors are used to reverse evolutions with minimal wall clock time. For the presented time-minimal parallel reversal schedules, the number l of physical steps that can be reversed grows exponentially as a function of the resource number $\varrho = c + p$. A corresponding software tool has been coded using MPI. Initial numerical tests are reported. They confirm the enormous reduction in memory requirement. Furthermore, the runtime behaviour is studied. It is verified that the wall clock time of the computation can be reduced compared to the logging-all approach if the memory access is comparatively costly. This fact is caused by the reduced storage in use. If the memory access is comparatively cheap, the theoretical runtime of time-minimal parallel reversal schedules is also confirmed.

The following overall conclusion can be drawn. For adjoining simulations $\log_{a(\bar{\tau})}(\#\text{ physical steps})$ processors and checkpoints are wall

clock equivalent to 1 processor and (# physical steps) checkpoints with $a(\bar{\tau}) = \frac{1}{2}(1 + \sqrt{1 + 4\bar{\tau}})$ and $\bar{\tau}$ the temporal complexity of a reverse step.

Acknowledgments

The authors are indebted to Daniele Casanova for his support during the numerical experiments and to Andreas Griewank for many fruitful discussions.

References

[1] J. Allen. Computer optimisation of cornering line. Master's thesis, School of Mechanical Engineering, Cranfield University, 1997.

[2] E. Bakker, H. Pacejka, and L. Lidner. A new tire model with an application in vehicle dynamics studies. *SAE-Paper*, 890087, 1989.

[3] Y. Evtushenko. Automatic differentiation viewed from optimal control. In G. F. Corliss and A. Griewank, editors, *Computational Differentiation: Techniques, Implementations, and Application*, Philadelphia, 1991. SIAM.

[4] M. Garey and D. Johnson. *Computers and intractability: Aguide to the theory of NP-completeness.* Freeman and Company, New York, 1980.

[5] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Frontiers in Applied Mathematics. SIAM, Philadelphia, 1999.

[6] 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.

[7] J. Grimm, L. Pottier, and N. Rostaing-Schmidt. 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*, Philadelphia, 1996. SIAM.

[8] W. Hager. Runge-kutta methods in optimal control and the transformed adjoint system. *Numer. Math.*, 87:247–282, 2000.

[9] P. Hilton and J. Petersen. A fresh look at old favourites: The fibonacci and lucas sequences revisited. *Australian Mathematical Society Gazette*, 25:146–160, 1998.

[10] U. Lehmann and A. Walther. The implementation and testing of time-minimal and resource-optimal parallel reversal schedules. Technical Report ZHR-IR-0109, Tech. Univ. Dresden, Center for High Perf. Comp., 2001.

[11] O. Talagrand. The use of adjoint equations in numerical modeling of the atmospheric circulation. In G. F. Corliss and A. Griewank, editors, *Computational Differentiation: Techniques, Implementations, and Application*, Philadelphia, 1991. SIAM.

[12] J. van de Snepscheut. *What computing is all about.* Texts and Monographs in Computer Science. Springer, Berlin, 1993.

[13] A. Walther. *Program Reversal Schedules for Single- and Multi-processor Machines.* PhD thesis, Tech. Univ. Dresden, Inst. for Sci. Comp., 1999.