Optimal program execution reversal

Andreas Griewank* Andrea Walther*

(Received 7 August 2000)

Abstract

For adjoint calculations, debugging, and similar purposes one may need to reverse the execution of a computer program. The simplest option of recording a complete execution log and then reading it backwards requires massive amounts of storage. Instead one may generate the execution log piecewise by restarting the "forward" calculation repeatedly from suitably placed checkpoints. Our goal is to minimize the temporal and spatial complexity as measured by the number of evaluation repeats and the number of checkpoints, respectively.

*Institute of Scientific Computing, Technical University Dresden, D-01062 Dresden, GERMANY. mailto:awalther@math.tu-dresden.de

[0]See http://anziamj.austms.org.au/V42/CTAC99/Grie for this article and ancillary services, Published 27 Nov 2000.

We present optimal checkpointing schedules for one-step and multi-step evolutions. These might arise for example as discretizations of ODEs by Euler's methods or multi-step schemes, respectively. Furthermore, we present parallel extensions, where auxiliary processors perform the repeated forward evaluations such that one processor can run backward without any interruption. For either case the length of the evolution that can be reversed is shown to grow exponentially with the number of checkpoints and either the number of repetitions or the number of processors.

Contents

1 Introduction and Assumptions

Our main motivation to consider program reversal schedules is the reverse mode of Algorithmic, or Computational, Differentiation [6]. It involves a forward sweep, where the trajectory of all intermediate states is usually stored, and a return sweep where this information is used to propagate adjoint states backward. The gradient of a scalar-valued function is yielded by the reverse mode 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.

However, the spatial complexity of the basic reverse mode is of the same order as the temporal complexity of the evaluation of the underlying function, since the full trajectory of intermediate results need to be recorded. Therefore, the practical use of the low temporal complexity of the reverse mode seems to be severely limited by the potentially excessive amount of memory required. Moreover, even if there is enough disc space to accommodatethe full trajectory, storing all intermediates on the way forward and

then retrieving them on the way back may slow the calculations considerably. Hence one may apply the checkpointing strategies developed below just to stay within main memory. This may well speed up the overall calculation, even though the operations count will be formally increased.

Algorithmic Differentiation (AD) is based on the observation that all function evaluations performed by a procedural language can be conceptionally viewed as a sequence of transformations

$$z_i = F_i(z_{i-1}) \quad \text{for} \quad i = 1, \ldots, l \quad \text{with} \quad z_i \in \mathrm{R}^n$$

where the components of z_i represent all values in the programs data set at stage i. In the simplest scenario only one scalar component is computed by each transformation. We will refer to the loop as an explicit time evolution on the state space R^n, starting from some initial $z_0 \in \mathrm{R}^n$. For simplicity, we will assume here that all F_i are at least once continuously differentiable at the arguments of interest. For what follows the individual transformations F_i need not be elementary at all but could represent subroutines whose evaluation involves millions of arithmetic operations. For example, in an experimental implementation of checkpointing within the overloading package ADOL-C [3] every ten thousand successive operations were combined to form one time step F_i. Also, the discretization of an initial value problem by Euler's method with fixed step size h leads to a recurrence whose time steps would most likely be rather uniform with regards to computational cost. As we will see such (near)-uniformity is quite desirable for our reversal purposes. Possibly for reasons of stiffness one may prefer to use implicit numerical integrators instead of explicit ones like Euler's method mentioned above. Since

they have to solve a nonlinear system of algebraic equations at each time step the complexity measures may then vary widely as a function of the number of iterations taken by the nonlinear solver in any particular time step. In any case we may assume that the temporal complexity of the overall function evaluation is roughly proportional to the number of time steps l.

Whereas we can cope with nonuniform step costs we always have to assume that the memory requirement for storing the intermediate states z_i may be considered constant. Otherwise the checkpoint persistence principle discussed in Section 4 does not apply. In practical application nonuniform state sizes might come about, for example, through adaptive grid refinements or through function evaluations that per se do not conform naturally to our notion of a sequence of transformations on a state space of fixed dimension. Assuming that the space size is constant we may measure the total memory requirement of a method in units of checkpoints, where one intermediate state can be stored.

The paper is organized as follows. In Section 2 we explain why program executions must be reversed for the efficient calculation of gradients and adjoints. In Section 3 we formulate the resulting serial and parallel reversal problem as a discrete optimization task. In Sections 4 and 5 we sketch the essential characteristics of optimal serial and parallel reversal schedules under certain assumptions on the problem parameters. The paper ends with a brief conclusion section.

2 Forward and Reverse Differentiation

Let us denote the composition of the individual transformations F_i as the function

$$F \equiv F_l \circ F_{l-1} \circ \cdots \circ F_2 \circ F_1 \ : \mathrm{R}^n \mapsto \mathrm{R}^n \, .$$

From the chain rule it follows directly that

$$A \equiv \frac{\partial F(z)}{\partial z} = A_l A_{l-1} \ldots A_2 A_1 \in \mathrm{R}^{n \times n}$$

where the A_i represent the individual Jacobians

$$A_i \equiv F_i'(z) \equiv \frac{\partial}{\partial z} F_i(z) \bigg|_{z=z_{i-1}} \in \mathrm{R}^{n \times n}$$

each of them evaluated at exactly the same point z as $F_i(z)$. Now suppose we are not interested in calculating the complete sensitivity information represented by A but only a directional derivative

$$\dot{z}_l = \frac{d}{dt} F(z_0 + t\, \dot{z}_0) \bigg|_{t=0} \, .$$

Again by the chain rule we have to evaluate $\dot{z}_i = A_i \, \dot{z}_{i-1}$ for $i = 1, \ldots, l$. The intermediate vectors represent the directional derivatives of the intermediate results with respect to the initial direction $\dot{z}_0$. In meteorology this process is referred to as evaluating the *tangent linear model* [9]. In AD the same

approach is simply called the *forward mode*. The final "tangent" $\dot{z}_l$ represents the sensitivity of all final state components with respect to variations of the initial state z_0 along the direction $\dot{z}_0$.

A dual way of obtaining sensitivity information about the overall transformation F is to select an adjoint vector $\bar{z}_l^T \in \mathrm{R}^n$ and then look for the corresponding gradient

$$\bar{z}_0 = \frac{\partial}{\partial z}\, \bar{z}_l\, F(z)\bigg|_{z=z_0} \in \mathrm{R}^n\,.$$

Again using the chain rule we find that $\bar{z}_{i-1} = \bar{z}_i\, A_i$ for $i = l, l-1, \ldots, 1$ has to be calculated. This process is called the *reverse mode* of AD because the Jacobians A_i enter into the product in the opposite order in which they are naturally generated.

Assuming that the given n-vectors $\dot{z}_0$ and $\bar{z}_l$ are dense and disallowing exact cancellations we find that in either product evaluation each nonzero entry of each A_i occurs exactly once as a factor in a multiplication, which is followed by an addition. Hence the operations count for computing the gradient $\bar{z}_0$ from $\bar{z}_l$ is exactly the same as that for computing the tangent $\dot{z}_l$ from $\dot{z}_0$. Moreover, when all F_i involve only one elementary function, one can see quite easily that this operations count is only a small multiple of that for evaluating F without any differentiation. Thus we obtain the truly amazing result that not only tangents but also gradients have essentially the same operations count as the underlying vector-function, irrespective of how many variables there are. This crucial observation is still not widely appreciated.

While the operations count of the reverse mode is surprisingly cheap there is a downside, namely the need to somehow produce the intermediate states z_i in backward order for the evaluation of the local Jacobians A_i. The most straightforward solution to this problem is to save the data required onto a global stack. When each F_i involves only one elementary function the number of double values that need to be stored on the stack is roughly equal to the operations count. The evaluation of a nontrivial function F can involve an enormous number of operations so that the size of the stack might grow to several giga- or even tera-bytes.

3 The Abstract Reversal Problem

From now on we will represent the intermediate states z_i simply by their counters $i \in \mathrm{N}$. Suppose at any stage of our calculation we have a finite set $Z \subset \mathrm{N}$ of states that are stored somewhere in memory. In the situation discussed above we can evaluate the state i whenever its predecessor $i-1$ is known and thus a member of the current Z. In view of multi-step methods for the numerical integration of ODEs we consider the more general situation where not just one but $q \geq 1$ immediate predecessors $[i-q, i-1] \equiv [i-q, i)$ must be contained in Z in order to allow the evaluation and thus the inclusion of state i into Z. We refer to this action as *advance to state* i and associate with it a certain cost τ_i. To get going at all we assume that the initial state set is the interval $Z_0 = [0, q)$. Here and throughout $[i, j]$ denotes a contiguous

range of integers and $[i, j) \equiv [i, j-1]$.

The only other important action we may take is the removal of the last and thus largest element $m = \max(Z)$ provided $b \geq 1$ of its intermediate predecessors are in Z. We call this action *return at* m and assume normally $b \leq q$. The inequality $b < q$ arises for example in adjoints of multi-step recurrences that depend only linearly on some of the previous states. The cost associated with individual step reversals will be denoted by $\bar{\tau}_i$. The aim of the calculation is to return back to the beginning. Hence we require that at the end always $Z = [0, b)$.

At any stage we may release elements from the state set Z in order to ensure that a certain uniform bound $q + c$ on its cardinality $|Z|$ is not exceeded. One may think of c as the number of checkpoints in "external" memory while q reside in "internal" memory to facilitate the current computation. In order to force proper returns we disallow release of the currently largest element of $m = \max(Z)$. Without loss of generality we may perform all releases immediately after the states concerned are used for the last time as arguments, i.e., as one of the q or b predecessors required in advances and returns, respectively.

3.1 Serial Reversal Schedules

Any $\tilde{Z}$ that can be obtained from a given Z by performing an advance or a return will be called a serial successor of Z. This relation will be denoted by

$Z \to \tilde{Z}$ and the associated cost $\mathrm{cost}(Z, \tilde{Z})$ must be equal either to τ_i or to $\bar{\tau}_i$. A chain of such successions connecting the initial $[0, q)$ to the final $[0, b)$ will be called a feasible serial reversal schedule

$$[0, q) \equiv Z_0 \to Z_1 \to \cdots \to Z_{k-1} \to Z_k \equiv [0, b)$$

with $l \equiv \max\{i \in Z_j \mid 0 \le j \le k\}$. The corresponding cost is naturally

$$t \equiv \sum_{j=0}^{k-1} \mathrm{cost}(Z_j, Z_{j+1}) .$$

Now we formulate the following optimization task.

For given q, b, c, l and the advancing costs τ_i for $i = 1, \ldots, l$ find a reversal schedule $Z_0 \to \cdots \to Z_k = [0, b)$ that minimizes t.	(1)

Here we have not mentioned the returning costs $\bar{\tau}_i$ because any optimal reversal schedule returns each step i for $i = l, l-1, \ldots, 1$ exactly once. Furthermore in any optimal reversal schedule the maximal element $m_j = \max(Z_j)$ is at first monotonically increasing towards its maximum $l \equiv \max(Z_{\mathrm{vertex}})$ and then it decreases (weakly) monotonically towards zero. Here we have labelled the first state distribution at which l is attained as vertex set Z_{vertex} and one may assume also, that its immediate successor is obtained by the return at l. An optimal reversal for the situation $l = 16$ and $c = 3$, $b = 1 = q$, with uniform step complexity $\tau_i = \bar{\tau}_i = 1$ is displayed in Figure 1.

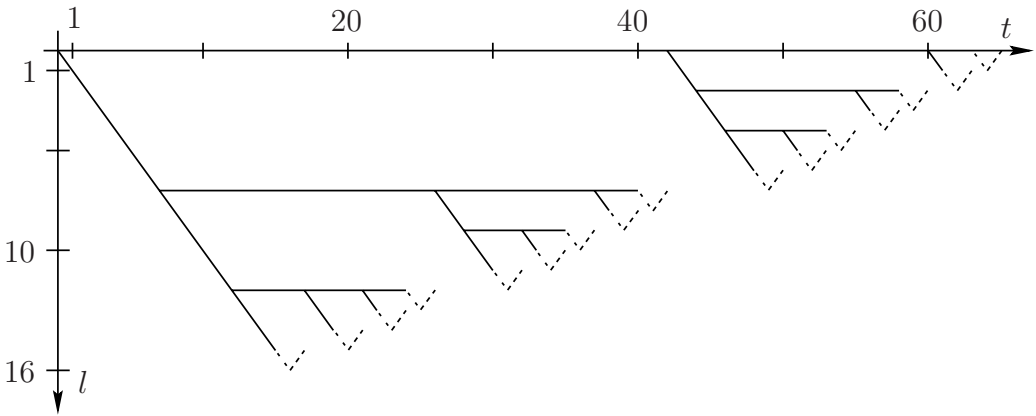

FIGURE 1: Optimal Serial Schedule for $c = 3$ and $l = 16$

The solid horizontal lines represent checkpoints and the slanted ones advances. The dotted hooks at the end represent the last advance and the subsequent return, both also requiring unit time by assumption. The vertex distribution is $Z_{\text{vertex}} = \{0, 7, 12, 16\}$, which here as always contains the maximal number of possible elements $c+q = 3+1 = 4$. It is also typical even for the parallel scenario considered later that the vertex distribution is reached in an uninterrupted forward sweep of l advances that leaves c intermediate states behind as checkpoints. The difficulty of the general combinatorial optimization problem (1) depends strongly on the parameters q, b, and whether the advance costs τ_i are uniform or not. The simplest case $q = b = 1$ and $\tau_i = \tau$ was considered already by Griewank [4] and Grimm et al. [7]. The

cases $q = b = 1$ with τ_i nonuniform or $1 \leq b \leq q$ with $\tau_i = \tau$ have recently been resolved quite satisfactorily in the thesis [10].

3.2 Parallel Reversal Schedules

When there is more than one processor available it seems natural to have them perform some of the repeated advances concurrently. In contrast the returns themselves cannot be parallelized, because they must happen in the prescribed reverse order. However, it is sometimes possible to perform certain preparatory calculations that do not depend on the adjoints $\bar{z}_{i+1}$ and in most cases one is forced to save certain intermediate quantities during the last advancing step to i. Therefore, this last advance to any particular i is sometimes called the *recording step* and its computational cost $\hat{\tau}_i$ may be significantly higher than τ_i. If there are sufficiently many auxiliary processors we can organize a just-in-time delivery of the intermediate states $i-1$. Some of the processors perform the recording steps to i such that one special processor can perform the returning steps at i for $i = l, \ldots, 1$ without any interruption. Thus the total runtime is given by

$$t = \sum_{i=1}^{l} \tau_i + \sum_{i=1}^{l} \bar{\tau}_i + \max_{j \leq l} \left(\hat{\tau}_j - \sum_{i=j}^{l} \tau_i \right). \tag{2}$$

The maximum in the last term will usually be equal to $\hat{\tau}_l - \tau_l$ so that the time is taken up by an advancing sweep to the state $l-1$, the recording

step to the final state l and a subsequent returning sweep from l down to 1. In particular this must be the case if we assume uniform step costs in that for three constants τ, $\bar{\tau}$, and $\hat{\tau}$ we have $\tau_i = \tau$, $\hat{\tau}_i = \hat{\tau}$ and $\bar{\tau}_i = \bar{\tau}$ for all i. Without this simplifying assumption it seems quite hard to coordinate the various advances such that they are performed concurrently by a limited number of processors. Then we may assign $\lceil \hat{\tau}/\bar{\tau} \rceil$ of them to do nothing but the recording steps and eliminate them from further consideration in the following sense.

In Section 5 we will discuss optimal parallel reversal schedules under the assumption $\hat{\tau} = \bar{\tau}$, in which case one dedicated processor can perform the recording steps in sync with the returning specialist already mentioned above. Then the minimal reversal time (2) has the simple value $l(\tau + \bar{\tau})$ and can be achieved as will be seen in Section 5. These same schedules may also be applied in the general uniform situation with $\lceil \hat{\tau}/\bar{\tau} - 1 \rceil$ extra processors dedicated to recording duties. As shown in [10] this approach is almost optimal in that l can only be increased by at most $\lceil \hat{\tau}/\bar{\tau} \rceil - 1$ if the extra processors are made available for other tasks before they settle down to their recording duties.

Hence we have essentially dispensed with the recording issue, except that we may realistically expect the crucial ratio $\bar{\tau}/\tau$ to be quite small when the recording steps perform a lot of preparatory simplifications. While fast return steps reduce the overall run-time $l(1+\bar{\tau}/\tau)\tau$ the no-interruption requirement becomes the harder to satisfy the smaller the ratio $\bar{\tau}/\tau$ is. Here harder means that the number ϱ of required processors or checkpoints becomes much larger.

Since for $\bar{\tau}/\tau \leq 1$ the reduction of $l(1 + \bar{\tau}/\tau)\tau$ is limited to a factor of 2 compared to $\bar{\tau}/\tau = 1$ it is perhaps acceptable that our theory caters as yet only for the cases where $\bar{\tau}/\tau$ is a positive integer. In other words we use besides q, b, and l, a fourth problem parameter $a \equiv \lceil \bar{\tau}/\tau \rceil$. In fact we have so far only worked through the one-step scenario $q = b = 1$ so that there are only the two problem parameters l and a in our current theory.

The cases $a = 1$ and $a = 2$ are probably most important for current applications. When $a = 1$ the uninterrupted backward sweep recedes exactly at the same speed as the original forward sweep, so that one may imagine a video tape running in reverse, at its normal speed. The situation $a = 2$ is typical for the evaluation of adjoints even through there is a variability in designing pairs of recording and returning steps.

It turns out that determining the maximal problem sizes $l \equiv l_\varrho$ for given $\varrho = p + c$ is both doable and sensible under the no-interruption condition. The fact that we throw the maximal number of processors p and checkpoints c together may seem at first a little surprising. The resulting *resource number* $\varrho = p + c$ represents a bound on the cardinality of the state distributions Z_j. Now each transition from Z_j to Z_{j+1} may involve up to p advances until we have reached Z_{vertex} and one advance less afterwards, because then one processor must be assigned to perform the uninterrupted returning run. To distinguish parallel from serial successors we write $Z_j \Rightarrow Z_{j+1}$ rather than $Z_j \rightarrow Z_{j+1}$ as before. For simplicity we use the shared memory model of parallelism. Hence we may formulate the general parallel reversal problem

formally as

$$\boxed{\begin{array}{l}\text{With } k \equiv l(1+a) \text{ for given } l,\ a,\ b, \text{ and } q \text{ find a parallel reversal}\\ \text{schedule } [0,q) = Z_0 \Rightarrow \cdots \Rightarrow Z_k = [0,b) \text{ that minimizes}\\ \varrho = p + c.\end{array}} \tag{3}$$

As we will see one can get by with only $p = \lceil(\varrho{+}1)/2\rceil$ advances per transition so that roughly half of the ϱ resources may actually be checkpoints rather than processors. Since l_ϱ grows always exponentially as a function of $\varrho = p{+}c$ the parameters $p \approx c \approx \varrho/2$ can stay in the single digits except for extremely large problem sizes l. Therefore, we may also assume that one checkpoint stays in the local memory of each processor rather than being shipped out to an external mass storage device. Given any size l with $l_{\varrho-1} < l \le l_\varrho$ we might apply the schedule for l_ϱ after suitably modifying its initial phase. Hence we have optimality with respect to the maximal number of resources used at any one time.

4 Optimal Serial Schedules

The key property that allows us to cut down the enormous variety of feasible reversal schedules is the principle of checkpoint persistence. In the one-step case $q = 1 = b$ we will call any element of Z other than the currently maximal state $m = \max(Z)$ and possibly its immediate predecessor $m{-}1$ a *checkpoint*, because it was shown in [6] that

Lemma 1 (Checkpoint Persistence) *Any reversal schedule can be modified without a reduction of the length l or an increase in the temporal cost t such that: once left behind by the release of their immediate successor $j+1$ all checkpoints j stay fixed until they are reversed. Moreover, during the "lifespan" of j between the advance to j and the return at j, all actions occur to the right, i.e. concern only states $k \geq j$.*

4.1 Schedule Computation by Dynamic Programming

Now suppose the overall reversal schedule is optimal in that it achieves the minimal time $t \equiv t(0, l, c)$. Then checkpoint persistence implies that the subschedule consisting of all actions to the right of a checkpoint j must also be time-optimal. Any reversal schedule for a subrange $[i, k]$ must keep the initial state i until its reversal at the very end of the calculation. Now let $part(i, k, c)$ denote the smallest number and thus the first intermediate state at which any reversal schedule that minimizes $t(i, k, c)$ sets a checkpoint. Then after the return to $j = \text{part}(i, k, c)$ the corresponding overall schedule must start the reversal of the remaining subrange $[i, j]$ from scratch but having the full set of c checkpoints available. Therefore we find that as an immediate consequence of the persistence principle

$$t(i, k, c) = \min_{i<j<k} \left\{ \sum_{s=i}^{j-1} \tau_{s+1} + t(j, k, c-1) + t(i, j, c) \right\}. \tag{4}$$

It is shown in [10] that the partition function has the monotonicity property

$$\text{part}(i, k-1, c) \leq \text{part}(i, k, c) \leq \text{part}(i+1, k, c)\,, \tag{5}$$

which can be used to limit the range of j over which the right hand side of (4) needs to be minimized to the interval with the bounds (5). Then it follows from the telescoping argument used by Knuth [8] in the context of binary search trees that $t(0, l, c)$ and an optimal reversal schedule can be computed in $O(c\,l^2)$ operations rather then the $O(c\,l^3)$ that we needed without exploitation of monotonicity.

The values of $\text{part}(i, k, c)$ specify unambiguously how optimal reversal schedules can be put together recursively from optimal schedules over subranges. Omitting the details of this transcription and noticing again that the return step times $\bar{\tau}_i$ enter only trivially into t we summarize

> Time optimal reversal schedules for one-step evolutions over l time steps with c checkpoints can be determined by dynamic programming in $O(cl^2)$ operations.

4.2 The Uniform One-Step Case

For the uniform one-step case it is possible to show in addition [7] that

Proposition 2 *When the step costs are uniform in that* $\tau_i = \tau$ *the cost* $t(i,k,c)$ *depends only on the integer pair* $l \equiv (k-i), c$ *and can be obtained explicitly as*

$$t(l,c) \equiv t(0,l,c) = l\,r - \beta(r,c+1)$$

where r *is the unique integer satisfying*

$$\beta(r-1,c) < l \le \beta(c,r) \quad \textit{with} \quad \beta(c,r) \equiv \binom{c+r}{r}$$

Using Stirlings formula one can show that asymptotically

$$\lim_{l\to\infty} \frac{t(l,c)}{l^{1+1/c}} = \sqrt[c]{c!} \approx \frac{c}{e}\,. \tag{6}$$

By comparison the spatial and temporal complexity of the basic version of the reverse mode are both of order l. 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.

4.3 The Multi-Step Case

The reversal schedules discussed above have been implemented in the software routine [5] and they can be generalized to the uniform multi-step situation $1 < q \ge b$, because the checkpoint persistence principle still applies.

Now checkpoints consist of q consecutive states $[q-i, i)$ that are needed to advance repeatedly towards i and beyond. When $b < q$ the extra linearity helps a little but does not effect the leading term in the temporal complexity. More specifically, we have as a consequence of Theorem 3.1 in [10]

$$\lim_{l\to\infty} \frac{t(l,b,q,c)}{l^{1+q/c}} = \left[\frac{(c/q)!}{q}\right]^{q/c} \approx \frac{c}{e\, q^{(1+q/c)}}\,. \tag{7}$$

This is exactly the same asymptotic complexity that can be achieved by the uniform one-step reversal schemes discussed above if they are adopted in the following way.

Suppose the $1+l$ original time steps are interpreted as $1+l/q$ *mega-time steps* between *mega-states* comprising q consecutive states $[qi, qi+q)$ for $i = 0, \ldots, l/q$. Here one may have to increase l formally to the next integer multiple of q. While the number of time steps is thus divided by q, the complexity of a mega-step is of course q times that of a normal step. The linearity properties signalled by $b < q$ are ignored. Since the dimension of the state space has also grown q-fold we have to assume that the number of checkpoints c is divisible by q so that we can keep up to $1+c/q$ mega-states in memory. Then replacing t by t/q, l by l/q and c by c/q in (6) yields exactly (7). Hence one may conclude that directly exploiting multi-step structure and special linearity properties is only worthwhile for comparatively small problems. Otherwise, interpreting multi-step evolutions as one-step evolutions on mega-states yields nearly optimal results.

5 Optimal Parallel Schedules

The construction of optimal parallel reversal schedules is in our experience a lot harder than the same task in the serial case. First we wrote an exhaustive search procedure for implicitly enumerating all possible reversal schedules given c and p with the aim of maximizing l. The most efficient implementation of this approach yielded maximal values of l for $c + p \leq 10$ on one node of the Origin 2000 in a few hours of computing time. The first observation was that the values of l for $p \geq c$ dependent only on the "resource number" $\varrho = p + c$. One such optimal schedule is displayed in Figure 2.

As before horizontal lines represent checkpoints and slanted lines represent running processes of which there may now be more than one in any transition. Not very surprisingly the resource usage reaches a maximum at and near the vertex, i.e., at the end of the forward and the beginning of the return sweep. During the warm-up and cool-down phase on the other hand some checkpoints and processors are idle, which happens quite often in parallel scheduling.

To detect the structure of efficient parallel schedules the exhaustive search program was made to find amongst the schedules reaching the maximal l a representative that was minimal with respect to the needed resources in each transition. Then recursive patterns could be discerned, which lead to a theoretical analysis whose principal results are sketched below. First we have the following generalization of Lemma 1 [10].

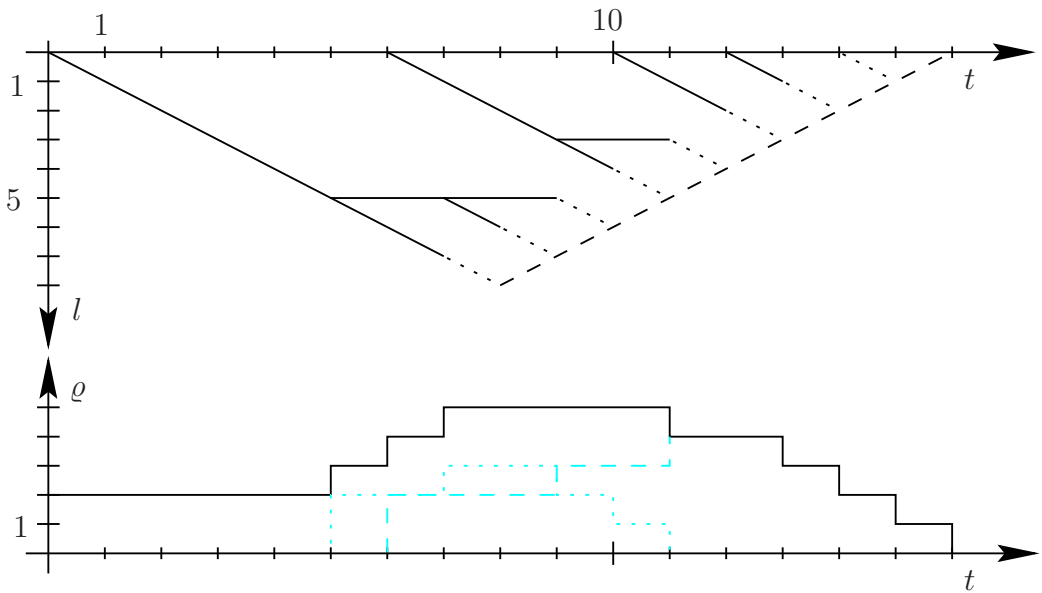

FIGURE 2: Optimal Parallel Reversal Schedule for $\varrho = 5$

Lemma 3 (Checkpoint and Processor Persistence) *Suppose there exists only a bound on $\varrho = c + p$, which means that checkpoints are convertible to processors and vice versa. Then any reversal schedule can be modified without loss of efficiency such that a checkpoint at a state i stays until i is returned, and an advancing process keeps running without interruption until the last state it reached is returned.*

Geometrically, we may interpret Lemma 3 as saying that the horizontal and slanted lines running to the right may bifurcate from each other but never merge or throw hooks until they reach the unique dashed line. An immediate consequence of Lemma 3 is that the first checkpoint set to a state $i > 0$ and the second process started from the initial state partition the schedule into a left and a right subschedule that include the initial state and the vertex, respectively. In Figure 2 this is true for the checkpoint at $i = 5$ and the process starting in transition Z_6. While this decomposition is reminiscent of the serial situation, the big difference is that now we have a region of overlap, where one has to ensure that the sum of the two (suitably shifted) resource profiles does not exceed the common bound ϱ. We did not succeed in casting the resource constraints in such a way that dynamic programming could be applied. Nevertheless, it is could be shown in [10] that

Proposition 4 *The maximal $l = l_\varrho$ fulfils the recursion $l_\varrho \leq l_{\varrho-1} + a\, l_{\varrho-2}$.*

The construction of reversal schedules that attain this upper bound is by no means trivial, especially since some of the subschedules are not optimal

themselves.

In the case $a = 1$ there exists a one parameter family of parallel schedules using ϱ resources whose lengths l_ϱ satisfy the Fibonacci recurrence $l_\varrho = l_{\varrho-1} + l_{\varrho-2}$ starting from $l_1 = 1$, $l_2 = 2$. For $a = 2$ another family of parallel schedules using ϱ resources exists whose length satisfy the generalized Fibonacci recurrence $l_\varrho = l_{\varrho-1} + 2\,l_{\varrho-2}$. Correspondingly the optimal schedules with ϱ resources can be decomposed into one using $\varrho - 1$ resources and two using $\varrho - 2$ resources. When $a > 2$ the recurrence $l_\varrho = l_{\varrho-1} + a\,l_{\varrho-2}$ could still be constructively verified, but again a decomposition into optimal subschedules could not be found.

At first one might think that demanding at least as many processors, namely p, as patches of memory for checkpoints, namely c is asking a little much. On the other hand it was found that having more that $p \approx \varrho/2$ processors does not help at all so that $p \approx c$ is indeed enough. Moreover, one may derive from the linear difference equation $l_\varrho = l_{\varrho-1} + a\,l_{\varrho-2}$ that asymptotically

$$l_\varrho \approx \left[1 + \sqrt{(1+4a)}/2\right]^\varrho ,$$

which means that $l = l_\varrho$ grows exponentially as a function of $\varrho \approx 2p$ and conversely that $p \approx c$ grows logarithmically as a function of l. For example we find that 8 processors and 8 checkpoint patches are sufficient to reverse 9104 time steps when $a = 1$ and 32768 time steps when $a = 2$. For $a > 3$ these values of l_ϱ are larger so that the case $p < c$ seems only important for machines with just two or four processors. Of course these subdiagonal

situations should also be resolved.

6 Conclusion

Schedules for reverting an evolution over l time steps within memory for c intermediate state vectors can be optimized on serial or parallel systems with respect to the total runtime t or the number of processors p, respectively. In the parallel scenario we demanded that the elapsed time attains its minimum, which requires that one processor can run backward without interruption. If c has significant size the serial runtime t or the number of processors p grow very slowly as a function on l. Conversely, l grows very rapidly as a function of l/t or p, respectively. Hence we can conclude that evolutions over very many time steps can be reverted at a reasonably cost in terms of memory and runtime or processors. Thus one can base parameter identification and design optimization on accurate gradient values even when the computer model in question involves simulations over very long periods of discrete or continuous time.

References

[1] Ch. Bennett. Logical reversibility of computation. *IBM J. Research and Development*, 17:525–532, 1973.

[2] A. Griewank and G.F. Corliss (eds.). *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, SIAM, Phil., 1991. C651, C652

[3] A. Griewank, D. Juedes, and J. Utke. ADOL-C, a package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Soft.*, 22:131–167, 1996. C630

[4] A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods and Software*, 1(1):35–54, 1992. C637

[5] A. Griewank and A. Walther. *Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation.* TOMS 26(1), 2000.

[6] A. Griewank. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation.* Frontiers in Appl. Math. 19, Phil., 2000. C644 C629, C641

[7] J. Grimm, L. Potter, and N. Rostaing-Schmidt. Optimal time and minimum space-time product for reversing a certain class of programs. In [2], pages 95–106. C637, C643

[8] D.E. Knuth. *The Art of Computer Programming.* Computer Science and Information Processing 3, Addison-Wesley, MI, 1976. C643

[9] O. Talagrand. The use of adjoint equations in numerical modelling of the atmospheric circulation. In [2], pages 169–180. C632

[10] A. Walther. *Program Reversal Schedules for Single- and Multi-processor Machines*. Ph.D. thesis, Inst. of Sci. Comp., TU Dresden, 1999.