Advantages of Binomial Checkpointing for Memory-reduced Adjoint Calculations

Andrea Walther[1] and Andreas Griewank[2]

[1] Institute of Scientific Computing, Technische Universität Dresden, Germany
awalther@math.tu-dresden.de
[2] Institute of Mathematics, Humboldt University Berlin, Germany
griewank@mathematik.hu-berlin.de

1 Introduction

For many time-dependent applications, the corresponding simulations can be performed using ordinary or partial differential equations. Furthermore, quite often there are quantities that influence the result of the simulation. Throughout, we assume that these quantities are control functions, for example heating in and/or at the boundary of a domain. To compute an approximation of the simulated process for a time interval $[0, T]$, one applies an appropriate integration scheme given by

$$\begin{aligned} y_0 &= y^0 \\ y_i &= F_i(y_{i-1}, u_{i-1}, t_{i-1}) \qquad i = 1, \ldots, N \ , \end{aligned}$$

where $y_i \in \mathbf{R}^n$ denotes the state and $u_i \in \mathbf{R}^m$ the control at time t_i for a given time grid $t_0, \ldots, t_N$ with $t_0 = 0$ and $t_N = T$. The operator $F_i : \mathbf{R}^n \times \mathbf{R}^m \times \mathbf{R} \mapsto \mathbf{R}^n$ defines the time step to compute the state at time t_i. Note that we do not assume a uniform grid. To optimize a specific criterion or to obtain a desired state, the cost functional

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

measures the quality of $y(u)$ and $u = (u_1, \ldots, u_N)$. Here, $y(u) = (y_1(u), \ldots, y_N(u))$ describes the dependence of the state y on the control u. For applying a calculus-based optimization method, one may use an adjoint integration

$$\begin{aligned} \bar{y}_N &= 0 \\ \bar{y}_{i-1} &= \bar{F}_i(\bar{y}_i, y_{i-1}, u_{i-1}, t_{i-1}) \qquad i = N, \ldots, 1 \ , \end{aligned} \tag{1}$$

motivated by the adjoint differential equation that belongs to the differential equation describing the state. Subsequently or concurrently, the desired derivative information $\hat{J}_u(u)$ can be reconstructed from $\bar{y} = (\bar{y}_0, \ldots, \bar{y}_N)$. The specific choice of the adjoint steps $\bar{F}_i$ depends on the forward integration and whether one prefers the continuous adjoint or the discrete adjoint formulation, see e.g. [7, 3, 5]. For the purpose of this paper, it is only important to note that the adjoint integration has to be performed backwards in time and that the complete forward trajectory $y = (y_0, \ldots, y_{N-1})$ is required. Hence, storing all states $(y_0, \ldots, y_{N-1})$ during the forward integration and reading them in reverse order during the adjoint integration forms one simple possibility to overcome this difficulty. Then the computing time for the adjoint calculation consists of the evaluation of N time steps F_i storing the state y_{i-1} and the evaluation of N adjoint steps $\bar{F}_i$.

The storage requirement of the basic approach to calculate adjoints is proportional to the number N of time steps. If we want to calculate the adjoint of a real-world problem with thousands of time steps this memory requirement of the basic approach may become a serious problem. For example, for computing 3D flows with unstructured grids one may need easily 10 to 100 MBytes to store only one state vector y_i [10]. Therefore, it is reasonable to assume that due to their size, only a very limited number of intermediate states can be kept in memory. They may serve as checkpoints, such that the required information for the backward integration is generated piecewise during the adjoint calculation. Sections 2 and 3 present two different checkpointing techniques. The resulting run-times and memory requirements are compared in Section 4. Finally, some conclusions and an outlook are given in Section 5.

2 Uniform Checkpoint Distribution

To distribute the checkpoints equidistantly over the given number of time steps forms one obvious solution to the storage requirement problem. Subsequently the adjoints are computed for each of the resulting groups of time steps separately. Denoting the number of checkpoints used by c, the corresponding calculation of the adjoint values can be performed using the following algorithm where the counter i is identified with the state y_i:

Two-level Checkpointing

Initialization: Reserve space for c_1 checkpoints, store the initial state y_0 in the first one and set

$$c_2 = \begin{cases} \lceil N/(c_1+1) \rceil & \text{if} \quad c_1 \lceil N/(c_1+1) \rceil < N \\ \lfloor N/(c_1+1) \rfloor & \text{else} \end{cases}$$

Advance: Starting from the initial state, advance to state $c_1 \cdot c_2$ by performing the time steps F_i, $1 \leq i \leq c_1 \cdot c_2$. While integrating forward, store the states $(j-1)\,c_2$ in the checkpoints j for $j = 2, \ldots, c_1$.
Reverse:
do $p = c_1$, 0, -1
  Evaluate the time steps F_i, $p \cdot c_2 < i < N$ storing the states i, $p \cdot c_2 \leq i < N-1$,
  perform the adjoint steps $\bar{F}_i$, $N \geq i > p \cdot c_2$ to calculate the adjoints,
  set $N = p \cdot c_2$, if $p > 0$ read the contents of checkpoint p.
end do

Fig. 1 sketches the two-level checkpointing for $N = 16$ time steps and $c = c_1 + c_2 = 6$. Throughout, the time steps are plotted along the vertical axis and the computing time required for the adjoint calculation is represented by the horizontal axis. Each solid horizontal line including the horizontal axis itself represents a checkpoint. The time, when a state is stored in a checkpoint, is marked with a black circle for the first level and with a black square for the second level. The slanted black lines represent the evaluation of time steps. The adjoint steps are drawn as dashed slanted lines. Finally, black arrows depict the usage of a state y_i for an adjoint step F_{i+1} without performing the corresponding time step F_i. This adjoint calculation is possible due to the assumed structure (1) of the adjoint steps. Note, that it may be required to evaluate F_N once to initialize the adjoints. This evaluation can be introduced right after the evaluation of F_{N-1} for $p = c_1$. For illustration purposes, we suppose throughout that all time steps and all adjoint steps have the same temporal complexity normalized to 1. However, to apply the presented

optimal checkpointing techniques, only the identical temporal complexity of all time steps is required. As can be seen, 24 time steps are performed in the example. Hence,

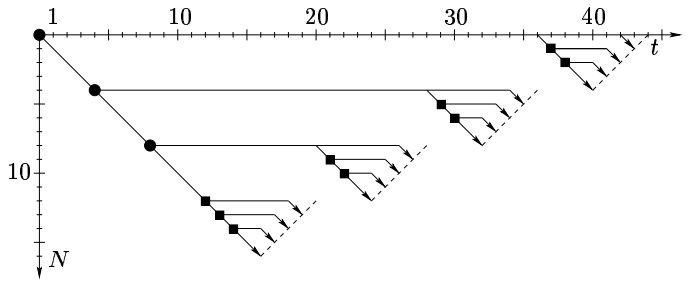

Fig. 1. Two-level checkpointing for $N = 16$ time steps and $c = c_1 + c_2 = 6$ checkpoints

the number of additional time step evaluations caused by the two-level checkpointing compared to the basic approach equals 9. Furthermore, at most 6 states have to be kept in memory.

The two-level checkpointing has been proposed several times in the literature, e.g. [11, 1], and is easy to implement. Naturally, one can apply two-level checkpointing repeatedly for the groups of time steps that are separated by equidistant checkpoints. This approach is called *multi-level checkpointing* [6] and sketched by Fig. 2 for the three-level case. The multi-level checkpointing is defined by the number of levels r, the number of checkpoints c_i that are uniformly distributed at level i, $i = 1, \ldots, r-1$, and the number of states c_r that have to be stored at the highest level r. Hence, the parameters of the adjoint calculation shown in Fig. 2 are $c_1 = 2$, $c_2 = 2$, and $c_3 = 1$. For a given r-level checkpointing, one easily derives the following identities

$$N_r = \prod_{i=1}^{r}(c_i + 1), \quad M_r = \sum_{i=1}^{r} c_i, \quad T_r = \sum_{i=1}^{r} \frac{c_i\, N_r}{c_i + 1} = r\, N - \sum_{i=1}^{r} \prod_{\substack{j=1 \\ j \neq i}}^{r} (c_j + 1)\ ,$$

where N_r denotes the number of time steps for which the adjoint can be calculated using the specific r-level checkpointing. The corresponding memory requirement equals M_r. The number of time step evaluations required for the adjoint calculation is given by T_r, since at the first level $c_1 N_r/(c_1 + 1)$ time steps have to be evaluated to reach the second level. At the second level, one group of time steps is divided into $c_2 + 1$ groups. Hence, $c_2(N_r/c_1 + 1)/(c_2 + 1)$ time steps have to be evaluated in each group to reach the third level. Therefore, we obtain $(c_1 + 1)c_2(N_r/c_1 + 1)/(c_2 + 1) = c_2 N_r/(c_2 + 1)$ at the second level and so on. It follows that each time step F_i is evaluated at most r times. Hence, if we apply two-level checkpointing, a specific time step is evaluated no more than two times.

The two- as well as the multi-level checkpointing technique have the drawback that at each level the checkpoints are not reused. Each checkpoint stores at each level only one state and becomes idle as soon as the data that is stored in the checkpoint has been used for the adjoint calculation. A method that reuses the checkpoints as soon as possible is proposed in the next section.

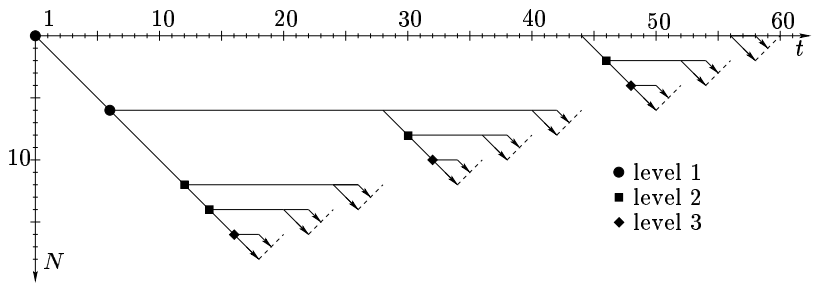

Fig. 2. Three-level checkpointing for $N = 18$ time steps and $c = 5$ checkpoints

3 Binomial Checkpoint Distribution

When one applies the checkpointing technique proposed in [5], the adjoint values are again generated piece by piece but only one state is employed for the adjoint calculation at any time. Therefore, the checkpointing procedure has to be adapted as follows:

Binomial Checkpointing

Initialization: Reserve space for c checkpoints and store the initial state y_0 in the first one.
do $p = N$, 1, -1
Advance: Starting from the last checkpoint assigned, advance to state $p - 1$.
If checkpoints are free, set as many of them as possible to states i along the way.
Reverse: Perform the adjoint step $\bar{F}_p$ to calculate the adjoint.
If state $p - 1$ is stored in a checkpoint, free the checkpoint up.
end do

The memory requirement of this checkpointing procedure equals $M_b = c$. Naturally, the question arises where one should place the checkpoints in the action "Advance" of the algorithm to minimize the number of time step evaluations. The application of the routine revolve ensures that the initiated checkpointing process is provably optimal with respect to the run-time increase for a given number of checkpoints [5]. More specifically, for the structure (1) of the adjoint steps considered here, the following complexity result holds:

Theorem 1. *Let N be the total number of time steps for which the adjoint has to be calculated. Suppose, up to c checkpoints are available at any time. Then the minimal number of time step evaluations needed for the adjoint calculation equals*

$$T_b = N\,r - \binom{c+r}{r-1},$$

where r the unique integer satisfying

$$\binom{c+r-1}{r-1} \;<\; N \;\leq\; \binom{c+r}{r}. \tag{2}$$

The proof of Theorem 1 as presented in [5] constructs recursively checkpointing schedules that attain the minimal number T_b. For the optimal checkpointing procedures the positions of the checkpoints are given by binomial coefficients. This fact explains the name *binomial checkpointing*. Furthermore, the proof of Theorem 1 shows that each time step

F_i is evaluated at most r times. Hence, r has the same meaning as in the previous section. It was proved earlier that a logarithmic growth of memory and run-time can be achieved using binomial checkpointing by providing an appropriate number of checkpoints [4].

The routine revolve implements the optimal binomial checkpointing and can be incorporated easily in an existing adjoint calculation [2, 8]. Moreover, one can build a heuristic based on revolve such that the adjoint calculation using binomial checkpointing becomes applicable also if the number of time steps is not known a-priori, e.g. due to adaptive time stepping, and/or if the temporal complexity of the time steps is not constant, e.g. due to implicit methods [9].

One optimal checkpointing schedule computed with revolve for $N = 16$ time steps and $c = 3$ checkpoints is shown in Fig. 3. Once more, it might be necessary to evaluate

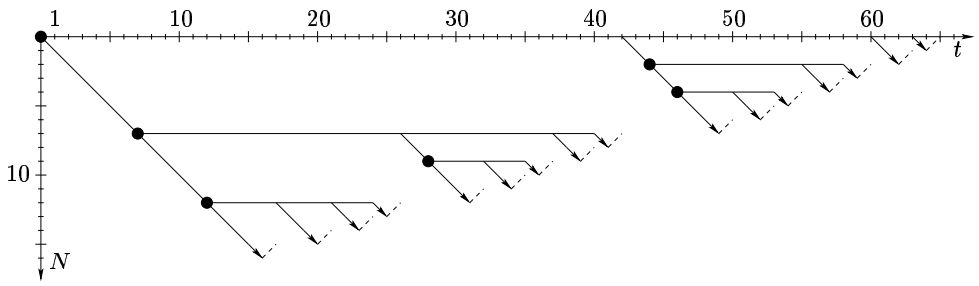

Fig. 3. Binomial Checkpointing for $N = 16$ time steps and $c = 3$ checkpoints

F_N once to initialize the adjoints. Since the situation is the same for the multi-level checkpointing and does not influence the results in the sequel, we ignore throughout the evaluation of F_N. For the example shown above, the number of time step evaluations equals $T_r = 33$. Compared to the two-level checkpointing, the computing time for the adjoint calculation increases by less than 50 %. Furthermore, only 3 states have to be kept in memory. Hence, the storage requirement is reduced by 50 %. The relation between the two checkpointing approaches will be discussed in more detail in the next section.

4 Comparison of Both Checkpoint Distributions

The integer r has the same meaning for both checkpointing approaches, namely the maximal number of times any particular time step F_i is evaluated during the adjoint calculation. Hence for comparing both approaches, assume at the beginning that r has the same value and that the same amount of memory is used, i.e. $M_r = M_b = c$.

Now, we examine the maximal number of time steps N^* for which an adjoint calculation can be performed using the two approaches. Assuming that r is a divisor of c and $M_r = c$, one obtains the identity

$$N_r^* = \left(\frac{c}{r} + 1\right)^r = \left(\frac{c+r}{r}\right)^r \qquad \text{with} \qquad c_i = \frac{c}{r}, \quad i = 1, \ldots, r$$

for the uniform checkpoint distribution because of the structure of N_r. For the binomial checkpoint distribution, Theorem 1 yields

$$N_b^* = \binom{c+r}{r} = \prod_{i=0}^{r-1} \left(\frac{c}{r-i} + 1\right).$$

Obviously, one has

$$\frac{c}{r} + 1 < \frac{c}{r-i} + 1 \qquad \text{for } 0 < i \leq r-1 \ .$$

These inequalities yield $N_r^* < N_b^*$ if $r \geq 2$. Hence for all $r \geq 2$ and and a given c, binomial checkpointing allows the adjoint calculation for a larger number of time steps compared to uniform checkpointing. In more detail, using Stirling's formula we obtain

$$\frac{N_b^*}{N_r^*} \approx \binom{c+r}{r} \left(\frac{c}{r}+1\right)^{-r} = \frac{1}{\sqrt{2\pi r}} \left(\frac{c}{r}+1\right)^{c} \approx \frac{1}{\sqrt{2\pi r}} \exp(r) \ .$$

Hence, the ratio of N_b^* and N_r^* grows exponentionally in r without any dependence on the number of available checkpoints. Fig. 4 shows N_r^* and N_b^* for the most important, moderate values $2 \leq r \leq 5$. Since r denotes the maximal number of times a specific

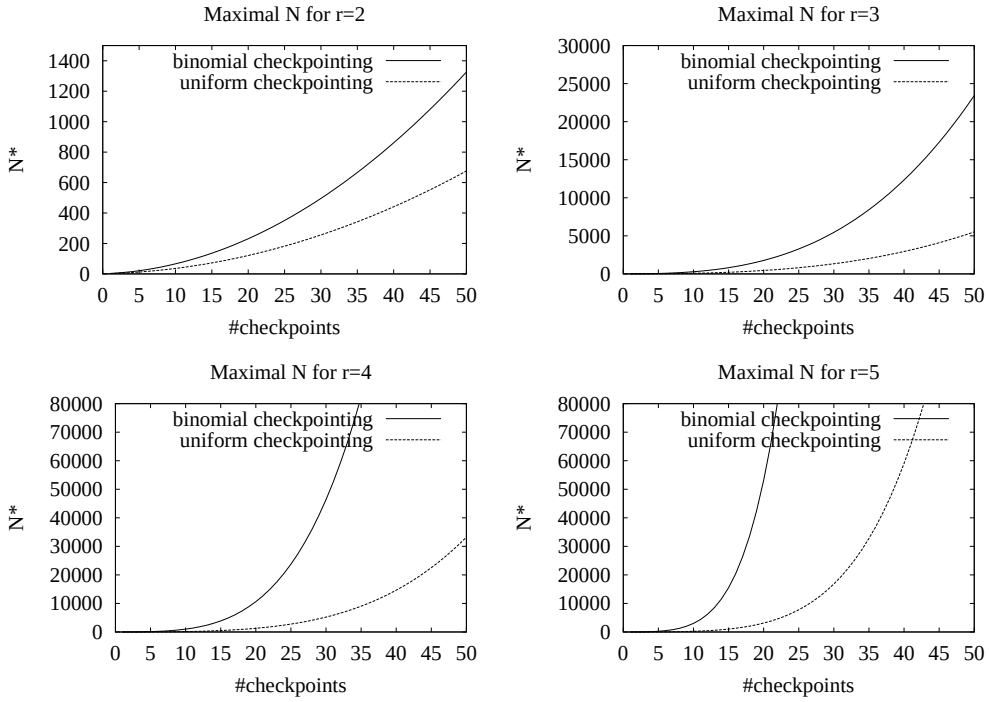

Fig. 4. N_r^* and N_b^* for $r = 2, 3, 4, 5$

time step is evaluated, we have the following upper bounds for the number of time steps evaluated during the adjoint calculation using r-level checkpointing and binomial checkpointing, respectively:

$$T_r = c\left(\frac{c}{r}+1\right)^{r-1} < r\, N_r^* \qquad \text{and} \qquad T_b = r N_b^* - \binom{c+r}{r-1} < r\, N_b^* \ .$$

For example, it is possible to compute the adjoint for $N = 23000$ time steps with only 50 checkpoints, less than $3N$ time step evaluations, and N adjoint steps using binary

checkpointing instead of three-level checkpointing, where $N_3^* \leq 5515$. If we allow $4N$ time step evaluations then 35 checkpoints suffice to compute the adjoint for 80000 time steps using binomial checkpointing, where $N_4^* \leq 9040$. These number are only two possible combinations taken from Fig. 4 to illustrate the really drastic decrease in memory requirement that can be achieved if binomial checkpointing is applied.

However, usually the situation is the other way round, i.e. one knows N and/or c and wants to compute the adjoint as cheap as possible in terms of computing time. Here, the first observation is that r-level checkpointing introduces an upper bound on the number of time steps the adjoint of which can be computed, because the inequality $N \leq (c/r+1)^r$ must hold. Furthermore, binomial checkpointing allows for numerous cases also a decrease in run-time compared to the uniform checkpointing. For a given r-level checkpointing and $M_r = c$, one has to compare T_r and T_b. Let r_b be the unique integer satisfying (2). Since at least one checkpoint has to be stored at each level, one obtains the bound $r \leq c$. I.e., one must have $c >= \log_2(N)$ to apply uniform checkpointing. Therefore, the following combinations of r and r_b are possible for the most important, moderate values of r:

$$r = 3 \Rightarrow r_b \in \{2,3\}, \qquad r = 4 \Rightarrow r_b \in \{3,4\}, \qquad r = 5 \Rightarrow r_b \in \{3,4,5\}\ .$$

For $3 \leq r \leq 5$, one easily checks that $T_r > T_b$ holds if $r_b < r$. For $r = r_b$, one can prove the following, more general result:

Theorem 2. *Suppose for a given N and a r-level checkpointing with $M_r = c$ that the corresponding r_b satisfying (2) coincide with r. Then, one has*

$$\begin{array}{ll} T_2 = 2N - c - 2 \ = \ T_b & \text{if } r = r_b = 2 \\ T_r > T_b & \text{if } r = r_b > 2. \end{array}$$

Proof: For $r_b = r = 2$ the identity $T_2 = T_b$ is clear. For $r = r_b > 2$, the inequality

$$\begin{aligned} \sum_{i=1}^{r} \prod_{\substack{j=1 \\ j\neq i}}^{r} (c_j + 1) &= \frac{(r-1)!}{(r-1)!} \left(\prod_{j=1}^{r-1} (c_j + 1) + (c_r + 1) \sum_{i=1}^{r-1} \prod_{\substack{j=1 \\ j\neq i}}^{r-1} (c_j + 1) \right) \\ &< \frac{1}{(r-1)!} \prod_{i=2}^{r} \left(\sum_{j=1}^{r} c_j + i \right) \ = \ \binom{c+r}{r-1} \end{aligned}$$

holds. Using the definitions of T_r and T_b, this relation yields immediately $T_r > T_b$. ∎

Hence, except for the case $r = r_b = 2$, where T_r and T_b coincide, the run-time caused by binomial checkpointing is less than the one caused by multi-level checkpointing if $r = r_b$.

5 Conclusions

This article discusses several checkpointing techniques, namely multi-level checkpointing and binomial checkpointing. A detailed analysis of the number of time steps the adjoint of which can be calculated, the run-time needed for the adjoint calculation and the memory requirement is given.

One can conclude that binomial checkpointing allows adjoint calculations with a surprisingly small fraction of the memory needed by the basic approach. This storage reduction causes only a very moderate increase in run-time. On the other hand, we see

that r-level checkpointing induces for a given number of checkpoints an upper bound on the number of time steps the adjoint of which can be computed. This upper bound can only be increased by introducing a next level of checkpointing. In addition it is shown that the run-time required for the adjoint calculation with r-level checkpointing exceeds the run-time needed for binomial checkpointing for the most important values of $r > 2$, whereas for $r = 2$ both methods yield the same run-time. However, for $r = 2$ and a given amount of memory, binomial checkpointing allows the adjoint computation for a larger number of time steps. Hence, even for $r = 2$ binomial checkpointing is preferable.

Moreover, it is quite often the case that the number N of time steps is not known a-priori, for example due to an adaptive time stepping method. Then, it becomes difficult to distribute the checkpoints for the two- or multi-level checkpointing such that the minimal run-time is attained. For binomial checkpointing the extension a-revolve deals with the unknown number of time steps by using a heuristic for the checkpoint placements. In addition, a-revolve can also handle time steps with varying temporal complexity. For time steps the cost of which do not change drastically, the heuristics implemented in a-revolve work well such that the corresponding adjoint calculation is only a few percentages slower than the one based on revolve [9]. Hence, binomial checkpointing provides memory-reduced adjoint calculation also in more general situations.

References

1. Charpentier, I. (2001): Checkpointing schemes for adjoint codes: Application to the meteorological model Meso-NH. SIAM Journal on Scientific Computing, **22**, 2135–2151
2. Gockenbach, M., Reynolds, D., Symes, W. (2002): Efficient and Automatic Implementation of the Adjoint State Method. ACM Transactions on Mathematical Software, **28**, 22–44
3. Griesse, R. (2003): Parametric Sensitivity Analysis in Optimal Control of a Reaction-Diffusion System—Part II: Practical Methods and Examples. Report No. 271, SFB F003, University of Graz, Austria. Submitted
4. Griewank, A. (1992): Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. Optimization Methods and Software, **1**, 35–54
5. Griewank, A., Walther, A. (2000): Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. ACM Transaction on Mathematical Software, **26**, 19–45
6. Heimbach, P., Hill, C., Giering, R. (2003): An efficient exact adjoint of the parallel MIT general circulation model, generated via automatic differentiation. To appear in Future Generation Computer Systems, 2003.
7. Hinze, M. (1999): Optimal and instantaneous control of the instationary Navier-Stokes equations. Habilitationsschrift, Fachbereich Mathematik, Technische Universität Berlin
8. Hinze, M., Walther, A. (2002): Discrete approximation schemes for reduced gradients and reduced Hessians in Navier-Stokes control utilizing an optimal memory-reduced procedure for calculating adjoints. Preprint MATH-NM-06-2002, Tech. Uni. Dresden. Submitted
9. Hinze, M., Sternberg, J. (2003): A-Revolve: An adaptive memory- and run-time-reduced procedure for calculating adjoints; with an application to the instationary Navier-Stokes system. Preprint MATH-NM-01-2003, Tech. Uni. Dresden. Submitted
10. Kaltenbach, H.-J., Jürgens, W., Spille, A. (2001): Numerische Simulation, Beeinflussung und Eigenmoden-Analyse einer abgelösten Strömung mit Querkomponente. In: Ergebnisberichte SFB 557 TP A6
11. Kubota, K. (1998): A Fortran77 preprocessor for reverse mode automatic differentiation with recursive checkpointing. Optimization Methods and Software, **10**, 319–335