ON RECENT DEVELOPMENTS FOR EFFICIENT TURBOMACHINERY DESIGN USING ALGORITHMIC DIFFERENTIATION

M. Sagebaum¹,*, J. Blühdorn¹, N.R. Gauger¹,
J. Backhaus², C. Frey², G. Geiser², F. Bayer², E. Kügeler²,
C. Battistoni³, M. Nagel³

¹ Chair for Scientific Computing, University of Kaiserslautern-Landau (RPTU), Germany
² Institute of Propulsion Technology, German Aerospace Center, Germany
³ MTU Aero Engines, Germany

* Corresponding author: max.sagebaum@scicomp.uni-kl.de

ABSTRACT
Over the last decade, an effort between MTU Aero Engines, DLR Cologne, and the Chair for Scientific Computing at the University of Kaiserslautern-Landau (RPTU) has been made to create an adjoint solver for DLR’s internal flow solver TRACE. This paper summarizes the strategies and techniques adopted to achieve a memory and CPU-efficient adjoint code. The derivatives required for setting up the adjoint are computed with operator overloading algorithmic differentiation (AD). For the handling of MPI and task-based parallelism, special libraries needed to be developed. In addition, a wrapper for the linear system solver Spliss is introduced. During the effort, the performance of the adjoint solution was closely monitored and improvements were introduced. The robustness of the method was further enhanced for off-design operating points, e.g. near the stall line. This paper illustrates the above techniques by means of computational results for turbomachinery configurations along with run time and memory measurements.

KEYWORDS
hybrid parallelization, performance optimization, algorithmic differentiation, adjoint approach, derivative computation

INTRODUCTION
For the analysis of turbomachinery flows, models of various fidelity and complexity are implemented in DLR’s turbomachinery simulation suite TRACE (Franke and Morsbach, 2018). The following model choices are typical for aerodynamic optimizations of turbomachinery. The equations are the compressible Reynolds-averaged Navier-Stokes (RANS) equations in a frame of reference rotating around the x-axis,

\[
\frac{\partial \mathbf{q}}{\partial t} + \text{div} (\mathbf{F}(\mathbf{q})) + \mathbf{S}(\mathbf{q}) = 0
\]

with

\[
\mathbf{q} = \begin{bmatrix} \rho \\ \rho U_x \\ \rho U_y \\ \rho U_z \\ \rho E \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \rho \mathbf{U} \\ \rho U_x \mathbf{U} + \rho e_x - \tau_x \\ \rho U_y \mathbf{U} + \rho e_y - \tau_y \\ \rho U_z \mathbf{U} + \rho e_z - \tau_z \\ \rho UH_{\text{tot}} - \tau \mathbf{U} - \mathbf{Q} \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} 0 \\ 0 \\ -2\rho \omega U_z - \rho |\omega|^2 r \mathbf{e}_r \\ 2\rho \omega U_y - \rho |\omega|^2 r \mathbf{e}_r \\ 0 \end{bmatrix}
\]
where \( q \) is the conservative state, and \( F \) are the fluxes with density \( \rho \), pressure \( p \), velocities \( U = (U_x, U_y, U_z) \), rothalpy \( H_{\text{tot}} = e + \frac{1}{2} (\|U\|^2 - (|\omega| r)^2) + \frac{p}{\rho} \), radius \( r \), and angular velocity \( \omega \). The internal energy \( e \) of the gas is mostly computed using the ideal gas assumption. Viscous stresses \( \tau \) are computed using Stokes’ hypotheses for Newtonian fluids. Due to the formulation in a rotating frame of reference, Coriolis and centrifugal forces appear as source terms. Rows of different rotational speeds or directions are coupled via mixing planes. Meshes can contain combinations of structured and unstructured domains where abutting interfaces are used to connect blocks with non-matching vertices on either side. Modeling of the Reynolds stresses can be achieved by a variety of turbulence models and variations thereof, where Wilcox’ \( k-\omega \) is the most often applied model. The transition from laminar to turbulent flows can be modeled through transition models, e.g. the \( \gamma \)-Re\( _{\theta} \)-model.

For the minimization of a target functional \( f(y, x) \), TRACE can be interpreted as a constraint and defined as a fixed point iterator \( G \) that has reached the fixed point \( y^* \) for which holds \( y^* = G(y^*, x) \). \( y \in \mathbb{R}^n \) is the state of TRACE, e.g. the conservative variables \( q \), and \( x \in \mathbb{R}^m \) defines the parameters for the minimization, e.g. the turbine blade surfaces.

As described in Sagebaum et al. (2017), the Lagrange approach can be used to formulate a fixed point iterator

\[
\lambda_+ = G_{\text{adj}}(y^*, x, \lambda) := \frac{\partial f}{\partial y}(y^*, x) + \frac{\partial G}{\partial y}^T(y^*, x)\lambda
\]

for the adjoint computation and the fixed point \( \lambda^* \) can be used to compute the total sensitivities with respect to the target functional

\[
\frac{df}{dx} = \frac{\partial f}{\partial x} + \frac{\partial G}{\partial x}^T\lambda^*.
\]

The fixed point iteration (3) is called the reverse accumulation approach and was first formulated by Christianson (1994). In the case of the TRACE code, it was decided that the derivatives of \( f \) and \( G \) should be computed with algorithmic differentiation (AD).

The theory of AD (Griewank and Walther, 2008) describes that any computer program can be differentiated by handling all elemental operations \( \phi_i : \mathbb{R}^{v_i} \rightarrow \mathbb{R} \) for \( i = 1 \ldots N \). Elemental operations are for example \( \times, +, \sin, \exp \), etc. and for large scale applications like TRACE, \( N \) can be in the region of \( 10^{12} \). If the elemental operations describe a mathematical function \( y = F(x) \), then the reverse mode of AD computes

\[
\bar{x} = \frac{dF}{dx}^T \bar{y}
\]

where \( \bar{y} \) is the seeding for the reverse AD mode and \( \bar{x} \) the result. The Jacobian \( \frac{dF}{dx} \) is never set up by AD. The computation is done by first recording information on a so-called tape (stack) for each elemental operator \( \phi_i \) during the primal computation. This can be primal values, Jacobians as well as identifiers for the bar values. Afterwards, the recorded information is interpreted in a reverse fashion which computes

\[
\bar{u} \leftrightarrow \frac{d\phi}{du}^T \bar{w}
\]

for all \( i = N \ldots 1 \).
The application of AD to TRACE was done by operator overloading (OO). Here, the computational type, e.g. `double` is replaced with one of the CoDiPack types, e.g. `codi::RealReverse` (Sagebaum et al., 2019). This allows CoDiPack to track all the elemental operators and record the specific information for the reverse interpretation. For parallelization techniques like MPI and thread parallelization special considerations are required with respect to AD and are reported in Section 1. The AD handling of the linear system solver Spliss (Krizikalla et al., 2021) is described afterwards in Section 2. The following Section 3 will look at the hot-spot analysis and discuss improvements to the AD approach. Finally, we report about the validation and performance results in Section 4.

1 PARALLELIZATION

1.1 MPI

In TRACE, MPI communication is employed for the subdomain coupling in the domain decomposition parallelization approach. Further communication patterns include the reduction of integral boundary values as well as the gathering and scattering of surface values for non-local boundary conditions. Communication requests can be blocking or non-blocking. The AD handling of the communication was initially done with AdjointMPI (Schanen et al., 2010) for the blocking MPI communication and then switched to a new development — MeDiPack\(^1\) — which then could handle also non-blocking communication and allowed a proper treatment of the custom type data buffers used in TRACE.

1.1.1 MeDiPack

The AD tool add-on MeDiPack is developed by the Chair for Scientific Computing, University of Kaiserslautern-Landau (RPTU), to provide full coverage of the MPI standard for AD tools. In general, most MPI routines do an assignment \(b = a\) where \(a\) and \(b\) are located on different processors. The reverse mode equation for an assignment is after Equation (6) \(\bar{a} = \bar{b}\). The first point is that instead of overwriting the communicated values, they need to be updated. The second point is that the information flow is reversed, e.g. a `MPI_Send` will become `MPI_Recv`. An overview of the appropriate reverse communication patterns is described by Utke et al. (2009).

The design aim of MeDiPack is to formalize the implementation of proper AD handling as much as possible. The challenge with the MPI framework is that a lot of concepts (e.g. `send/recv` buffers, blocking, non-blocking, reductions, etc.) are used in several functions and the interaction of the concepts requires always a slightly different code. It is therefore quite hard to provide a general set of functions that can be used to implement the general handling of a MPI function. For example, `MPI_Send` and `MPI_Isend` require almost the same code, only that for `MPI_Isend` the function needs to be split into two parts. To properly handle the interaction of the MPI concepts, a code generation approach is used for the implementation of MeDiPack.

The general design of MeDiPack is split into three layers. The first one provides the modified MPI routines such that the AD handling is applied. The second layer provides an abstraction for the data types used in MPI with additional functionality that can be used to iterate over all AD types in the buffer. The third layer is an interface to the AD tool and provides the functionality for handling and modifying the AD types in the data buffers.

\(^1\)https://www.scicomp.uni-kl.de/software/medi/
1.1.2 Custom data type handling

The MPI routines in TRACE are designed such that they only work with byte buffers. The data is packed into and unpacked from the byte buffers via custom function calls as depicted in Figure 1 in black. Therefore no type information is available for the internal data layout of the byte buffers which is problematic for AD. By providing a TRACE specific implementation of the datatype interface from the second layer of MeDiPack, a custom TRACE specific MPI type for the byte buffers of TRACE could be created. Figure 1 shows this specific handling in red. Each MPI buffer in TRACE is extended with a list of the AD type locations. These locations are extracted during the packing of the data which are then used by MeDiPack. Before the regular communication, a shadow MPI communication is done for the AD locations in the buffer. Therefore, MeDiPack has the AD locations available when the byte buffer is received and can perform the AD specific handling.

1.2 Task-based shared memory parallelization

TRACE has been extended to use task-based parallelism on CPUs to improve its intra-node performance. This allows to handle load imbalances in a more flexible way. Possible causes for such imbalances are, e.g. the high computational cost of certain boundary conditions that TRACE’s grid partitioner (Geiser and Schröder (2020)) cannot yet fully take into account. Other possibilities are computationally expensive conditional functions like the implicit system’s Jacobian matrix that is reevaluated depending on the solution state in the single grid blocks. Also adaptive mesh refinement can cause large load imbalances and task parallel approaches can alleviate them (Meng et al. (2010)).

Iterative solvers like TRACE’s implicit solution algorithm generally can be expressed as a cyclic graph of interdependent tasks. Figure 2 shows an example for a simple Poisson solver. Solid arrows indicate dependencies in between the tasks inside one solver iteration while dashed arrows point into the next solver iteration. Notice that most tasks are instantiated for each grid block while other tasks, e.g. for reductions, only exist once. The blue frame depicts a conditional sub-graph that processes either the red or the green branch depending on its entry condition. Here, it is used to buffer output data every few iterations and write it asynchronously.

The dependencies in between the tasks enables a scheduler to identify tasks that are ready for being executed. Most tasks are affine to specific compute units to ensure shortest NUMA distances, which is critical for high performance. However, idle compute units can also process non-affine tasks according to a work-stealing policy that minimises NUMA distance as well. Such a dynamic control flow allows to perform independent computations while a regular static control flow might stall at synchronisation points. Therefore, the task scheduler closely interacts with TRACE’s communication interface such that a task will pause itself and yield while waiting for remote data. The scheduler will automatically pick an independent ready task and continue the paused task once the required remote data has been received.

It turned out that, while being a natural choice, OpenMP task parallelism is inadequate for TRACE. All tested runtime implementations suffered from problems with handling NUMA
affinities correctly and did not allow to yield tasks and resume them on different compute units. Therefore, a scheduler based on POSIX threads with the required functionality has been implemented.

The AD approach has been extended to cover this additional layer of shared memory parallelism. A thread-safe version of CoDiPack allows simultaneous recordings in different threads and safeguards internal shared data structures.

A task graph provides a clear view of the dependencies between independent sequential computational tasks. The reverse mode of AD reverses these dependencies. Therefore, if we flip the edges in the task graph and replace every primal task by the interpretation of its recordings, we obtain a natural representation of a parallel derivative computation. Constructs such as conditional subgraphs and the startup phase of the reverse graph require special handling.

In TRACE, communication requests are additional dependencies in the task graph. We therefore transform TRACE’s communication requests into corresponding reverse requests that are integrated into the reverse task graph such that all dependencies are covered. Internally, the reverse MPI handles provided by MeDiPack are not recorded onto the tape but attached to the appropriate reverse task entry.

Expressing the derivative computation within the same framework that is used for the primal parallelization offers interesting opportunities for AD. All features of the tasking framework such as scheduling flexibility and load balancing become available for the derivative computation. For example, we could use different numbers of threads for the primal and derivative computations. Furthermore, the task graph expresses ownership relations between data and tasks, e. g., for the two blocks in Figure 2. This could be leveraged to guarantee exclusive read access for a major part of the variables and hence overcome limitations due to atomic updates on adjoint variables (Blühdorn et al. (2023); Kaler et al. (2021)).

We observed promising performance in the Poisson solver example. We still investigate some performance aspects of the tasking framework and its application to TRACE is work in progress. Therefore, the tests in Section 4 do not make use of the tasking capabilities.
2 LINEAR SYSTEM HANDLING

The solution of the primal equations (c.f Eq. 1) are carried out by implicit pseudo-time marching. For that, a sparse linear system has to be solved in each iteration. TRACE features an internal solver for linear system of equations, which has recently been accompanied by a dedicated library called Spliss (Krzikalla et al., 2021). Spliss is specialized on equation systems that arise during CFD simulations, i.e. sparse matrices of dense sub-blocks. It combines a high degree of flexibility in the solution methods and matrix representations with the efficient use of modern high-performance computing hardware, e.g. distributed and shared memory parallelization as well as support for computational accelerators like GPUs through the use of modern C++. Supported solution methods include, among others, matrix splitting methods, Krylov subspace methods, line inversion, and multigrid. In the following section, we describe the differentiation of the Spliss library, which combines algorithmic differentiation with specialized handling of linear operations.

2.1 AD handling of Spliss

In general, the handling of linear systems \( Ax = b \) for AD is described by the equations

\[
\begin{align*}
  s &= A^{-1T} \bar{x}, \\
  \bar{W} &= -x \cdot s^T, \\
  \bar{b} &= + s
\end{align*}
\]

It is therefore mandatory that AD has access to the transposed matrix \( A^T \) as well as to the identifiers of the vectors \( b, x \), and the matrix \( W \). The usual approach of exchanging the floating point type in Spliss with the AD type and then extracting the required information would work but has the disadvantage that all data needs to be copied several times. In addition, the performance would decrease by a large factor because of the additional data per floating point type. It would also no longer be possible to use the GPU capabilities of Spliss.

The general idea for the AD handling of Spliss is that the data of all Spliss structures are split upfront such that, e.g., \( A \) and the identifiers for \( \bar{A} \) are stored in separate data structures. This is achieved by specializing the MatrixDataInterface in Spliss. It contains pointers to the real part of the matrix and a matrix with the AD specific identifier data. If a user requests a matrix block from the interface, a pseudo block is returned which encapsulates the other two blocks. This is depicted in Figure 3. In addition, the wrapper provides the facilities for the automatic generation of the transposed matrix. Each time a block is written, the transposed block is updated as well. If the transposed block is on a different MPI rank, then the update of the transpose is delayed until the linear system is actually solved.

3 PERFORMANCE ANALYSIS AND IMPROVEMENT

3.1 Hot-spot analysis

The general application of operator overloading AD generates a solution for the derivative calculation which can be maintained with a low effort and is always consistent with the primal
computation. In general, this solution is not optimal from a memory viewpoint. There are usually some hot-spots that generate most of the stored data on the AD tape. Therefore, analysis and handling of these hot-spots are usually mandatory for large scale applications.

For CoDiPack there is a special type that gathers memory and performance relevant AD data on a function level. This data can be represented as a tree where an example is shown in Figure 4. How such a hot-spot can be improved depends on the underlying algorithm of the code location. If the computations are not relevant for the derivatives, their recording can be simply ignored. Otherwise, the preaccumulation technique is used for TRACE.

The preaccumulation technique is defined by Griewank and Walther (2008, page 220). It assumes that for a function \( y = h(x) \) it holds that \( \text{MEM} \left( \frac{dh}{dx} \right) \) is smaller than \( \text{MEM}(\text{tape}(h(x))) \), that is, it requires less memory to store the Jacobian matrix of \( h \) on the tape than storing the data for the elemental function evaluated in \( h \). Iterative procedures like the Newton method are usually good examples of such algorithms. Preaccumulation is usually applied by first recording the evaluation of \( h(x) \) on the tape and then using the tape evaluation to compute the Jacobian matrix. Afterwards, the recorded data of \( h(x) \) is removed from the tape and the Jacobian is stored instead.

For certain applications, the sensitivity with respect to the preconditioner of a linear system can be ignored. This is described by Griewank and Walther (2008, page 383). A validation for the TRACE suite showed that this is usually the case and therefore the option to disable these dependencies is added. This improves the speed of the linear system handling and reduces the tape memory since the computations for the preconditioner do not need to be recorded.

### 3.2 RPM Overview

Some configurations (for example off design) are naturally unstable as the primal solution in steady state fails to represent the unsteadiness of the flow. As a result, a stabilization method is applied on the adjoint procedure: the recursive projection method (RPM) introduced in (Renac, 2011; Shroff and Keller, 1993). The method can be applied to a fixed-point procedure like \( G_{\text{Adj}} \) as defined in (3). It assumes, that there are \( d \in N \) eigenvalues \( \zeta_1, \ldots, \zeta_d \) which are greater or equal to 1. The corresponding eigenvalues span the divergent subspace \( \mathbb{P} = \text{span} \{ e_1, \ldots, e_d \} \) and the orthogonal complement \( \mathbb{Q} = \mathbb{P}^\perp \) represents the convergent subspace. The space of the adjoint solution \( \lambda \in \mathbb{R}^n \) can now be separated into \( \lambda = \lambda_p + \lambda_q \) with \( \lambda_p \in \mathbb{P} \) and \( \lambda_q \in \mathbb{Q} \). The
RPM method applied to $G_{Adj}$ reads now

$$\lambda_{q}^{l+1} = P_q G_{adj}(y^*, x, \lambda^l), \quad (7)$$

$$\lambda_{p}^{l+1} = \lambda_p^l + \left( I - P_p \frac{\partial G^T}{\partial y} (y^*, x) P_p \right)^{-1} (P_p G_{adj}(y^*, x, \lambda^l) - \lambda_p^l), \quad (8)$$

$$\lambda_{p}^{l+1} = \lambda_q^l + \lambda_p^l, \quad (9)$$

where $P_p$ and $P_q$ are the projectors onto the subspaces $\mathcal{P}$ and $\mathcal{Q}$, respectively. The projection of the linearized adjoint operator $\frac{\partial G^T}{\partial y}$ can easily be computed with AD.

The challenge in applying the RPM method lies in the creation of the subspace $\mathcal{P}$. The common approach is to use the Krylov criterion as described in Renac (2011). Here, the last $k$ iterates of $\lambda$ are stored and a QR-factorization is applied on the deltas of the iterates. The columns of the $Q$ matrix in the factorization are then considered as a basis for the divergent subspace $\mathcal{P}$. The convergence within the subspace $\mathcal{P}$ is demonstrated by Shroff and Keller (1993) as long as the primal solution $y^*$ is close to the real primal solution.

Off-design points are a prime target for stabilization methods. Indeed, the primal solution may have difficulties converging e.g. entering a limit cycle. Each step in this limit cycle may have different problematic eigenvalues. The fixed-point adjoint procedure $G_{adj}$ fixes one step in this limit cycle and so a problematic eigenvalue. In that case, the adjoint will inherit the instabilities and potentially diverge. In Fig.5, the adjoint solution immediately diverges. With the RPM method, it successfully detects the diverging eigenvectors and passes them into the subspace $\mathcal{P}$. This test case is a low pressure turbine test case (4 million cells, two stages with transition models) and is particularly unstable. One recommendation made by Shroff and Keller (1993) is to make a second projection from the eigenvector basis $V$ with $\frac{\partial G^T}{\partial y}$,

$$V \leftarrow QR_{\text{dec}} \left( \frac{\partial G^T}{\partial y} V \right), \quad (10)$$

$QR_{\text{dec}}$ is here meant as the Gram-Schmidt orthogonalization. For large test cases, the vectors $V$ that were supposed to form an orthonormal basis were in fact not orthogonal. The use of the augmented Gram-Schmidt allows to reduce the numerical error in these cases. Moreover, it has been found that the initial assumption that the solution diverges when the Krylov criterion is met is not sufficient. After most of the diverging eigenvectors are found, the method is still able to catch some eigenvectors that slow down the convergence rate. For some, although the Krylov criterion was satisfied, the solution suddenly worsened after including them in $\mathcal{P}$. We propose to add an additional criterion which considers a certain threshold in the diagonal values of the $R$ matrix in the QR decomposition. This prevents adding eigenvectors to $\mathcal{P}$ where the eigenvalue is too small. Two cases arise: either it is diverging and the Krylov jump is delayed until this criterion is satisfied. Or it is already converging and the Krylov jump may not happen at all which would not impair the convergence.

4 RESULTS

The considered test case is an eight stages transonic high pressure compressor with an intermediate guide vane (IGV) at the entry as well as the first three stators being variables guide vanes (VGV). A schematic view is given in Figure 6. The annulus is fixed and the optimization is realized on the components up to stage 4. The components are parametrically defined
forming a design space of more than 500 variables. Some mechanical parameters determine the angle of the IGV and VGVs but the majority define 2D parametric shapes at multiple radii for each airfoil of interest, the 3D shape is then interpolated via polynomials which generate a fully structured mesh. The steady state Navier-Stokes equations are solved with TRACE and the turbulence model is the $k$-$\omega$ model of Wilcox (1988). The optimization is made on 4 different working points shown schematically in Figure 7. The number of cells is adjusted ranging from 5 to 9 millions for partial load and full load, respectively. Objectives are the increase in efficiency at full load while constraining the solution field with regard to absolute pressure ratio and mass flow. Each new design iteration is calculated based on a gradient step method. The computation of the gradient for every objective and constraint is performed with the presented adjoint method in TRACE. More details on the settings of this optimization can be found in the thesis of Mann (2022).

4.1 Validation

The adjoint validation is carried out for a design solution from a full workload line point in Figure 7. In Figure 8, the residuals of this point are shown with regard to the efficiency objective. The residuals for the constraint on the mass flow have been omitted. For the efficiency objective, the primal deltas (finite difference gradient) (in red) are computed for various step sizes in each parameter direction. The projected adjoint values (in blue) are computed only for $h = 0$. Around $h = 0$ the gradient of each method (finite difference in red, and adjoint in blue) are in agreement with each other, thereby validating the adjoint procedures in TRACE. Some parameters show linear behaviors and some non-linear with regards to the objectives. Especially
for $\alpha_1$, the projected adjoint in $h = 0$ shows a good agreement that would not be captured with a finite difference method with a step size of e.g. $h = -1$.

### 4.2 Performance measurements

For the results we present, there are several different configurations available. The first distinction is made between the internal linear solver from TRACE and the Spliss library, they are called *Normal* and *Spliss*, respectively. Currently, the linear system solvers use either one sweep or two sweeps to partially solve the linear system. In addition, we compare two different AD type implementations from the CoDiPack AD tool. The *codi::RealReverse* type features a linear identifier management and the *codi::RealReverseIndex* type features a reuse identifier management scheme, they are called *Linear* and *Reuse*, respectively.

Table 1 shows the results for selected combinations. Since we are using a reverse accumulation approach, the reversal time is the important time factor for the adjoint computation. Nevertheless, the recording times stay within reasonable limits. The reversal time itself has a very good time factor of about 2.6 with the internal linear system solver and one sweep. The factor improves to 1.6 when switching to the AD handled Spliss linear system solver. The timing for two linear system sweeps increases slightly for the internal solver, with Spliss the timing increases quite drastically, which needs to be investigated. The memory factor of about 10.5
Table 1: Performance data and factors for one step of adjoint TRACE. Factors are with respect to the primal computation for one linear solver sweep (0.62 s, 65.13 GiB) and two sweeps (1.08 s, 65.13 GiB).

<table>
<thead>
<tr>
<th>Kind</th>
<th>Record time in s</th>
<th>Reversal time in s</th>
<th>Stabilization time in s</th>
<th>Total mem in GiB</th>
<th>Adjoint vec. in GiB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Normal 1 sweep</td>
<td>5.57 (8.98)</td>
<td>1.59 (2.56)</td>
<td>0.19 (0.31)</td>
<td>676.85 (10.39)</td>
<td>84.32 (1.29)</td>
</tr>
<tr>
<td>Linear Spliss 1 sweep</td>
<td>5.61 (9.05)</td>
<td>0.96 (1.56)</td>
<td>0.12 (0.19)</td>
<td>538.20 (8.26)</td>
<td>53.45 (0.82)</td>
</tr>
<tr>
<td>Reuse Normal 1 sweep</td>
<td>8.55 (13.79)</td>
<td>1.81 (2.92)</td>
<td>0.22 (0.36)</td>
<td>661.57 (10.16)</td>
<td>24.86 (0.38)</td>
</tr>
<tr>
<td>Linear Normal 2 sweeps</td>
<td>6.15 (5.70)</td>
<td>2.25 (2.09)</td>
<td>0.18 (0.17)</td>
<td>885.70 (13.60)</td>
<td>127.34 (1.96)</td>
</tr>
<tr>
<td>Linear Spliss 2 sweeps</td>
<td>8.10 (7.50)</td>
<td>3.61 (3.34)</td>
<td>0.13 (0.12)</td>
<td>558.57 (8.58)</td>
<td>53.45 (0.82)</td>
</tr>
<tr>
<td>Reuse Normal 2 sweeps</td>
<td>9.50 (8.80)</td>
<td>2.43 (2.25)</td>
<td>0.22 (0.20)</td>
<td>819.44 (12.58)</td>
<td>25.32 (0.39)</td>
</tr>
</tbody>
</table>

with respect to the primal memory consumption is within the reasonable margins for operator overloading AD solutions. It improves to 8.3 when the Spliss linear system solver is used. For the two sweeps, the memory factor for the internal solver increases from 10.4 to 13.6, which is not seen for the Spliss version. Here, the special handling of Spliss for AD prevents an additional memory consumption.

The time and memory factors for the two different CoDiPack types do not vary that much. The most important change is the size of the adjoint vector which comes into play when the adjoints for multiple target functionals \( f \) are computed at the same time since the size of the adjoint vector is allocated for each adjoint direction. If four functionals are computed at the same time, then the reversal time factor is 4.32, which is an improvement of 55% compared to an individual computation of all four adjoints. The adjoint vector memory for the linear CoDiPack type increases from 84.3 GiB to 337.3 GiB, and for the reuse CoDiPack type from 24.9 GiB to 99.4 GiB, which shows the improved memory scaling of the reuse index type.

**CONCLUSIONS**

We demonstrated the creation of an adjoint solver of TRACE with the help of AD. Advanced features like MPI communication and task parallelism did not prevent the application of AD but required the development of additional libraries for proper handling. CoDiPack and MeDiPack are openly available. For a good performance of the adjoint solver, the necessary analysis tools have been created and applied to TRACE. The detected hot-spots with respect to AD have been handled with appropriate techniques like preaccumulation, while specialized wrappers are implemented in case of the Spliss linear system solver. For off-design points and instabilities due to the numerical scheme, the adjoint method is enhanced with the recursive projection method such that the overall robustness is increased. The resulting time factor of 0.96 and memory factor of 8.26 for the adjoint solver are in the theoretical margins of operator overloading AD. The good performance results and the consistency of the adjoint computations with the current and future feature set of TRACE demonstrate the benefits of the AD approach.

**REFERENCES**


Franke, M. and Morsbach, C. (2018). Assessment of Scale-Resolving Simulations for Turbo-
machinery Applications. In Progress in Hybrid RANS-LES Modelling, volume 137 of Notes


Kaler, T., Schardl, T. B., Xie, B., Leiserson, C. E., Chen, J., Pareja, A., and Kollias, G.
(2021). PARAD: a work-efficient parallel algorithm for reverse-mode automatic different-
tiation. In Symposium on Algorithmic Principles of Computer Systems (APOCS), pages 144–
158, Philadelphia, PA, USA. SIAM.

Linear System Solver for Transparent Integration of Emerging HPC Technologies into CFD
Solvers and Applications. In New Results in Numerical and Experimental Fluid Mechanics

Mann, S. (2022). Aerodynamische Mehrpunktoptimierung eines Hochdruckverdichters mithilfe

Meng, Q., Luitjens, J., and Berzins, M. (2010). Dynamic task scheduling for the uintah frame-
work. In 2010 3rd Workshop on Many-Task Computing on Grids and Supercomputers, pages
1–10.

Renac, F. (2011). Improvement of the recursive projection method for linear iterative scheme
stabilization based on an approximate eigenvalue problem. Journal of Computational


Sagebaum, M., Özkaya, E., Gauger, N. R., Backhaus, J., Frey, C., Mann, S., and Nagel, M.
(2017). Efficient algorithmic differentiation techniques for turbo-machinery design. AIAA
2017-3998.


adjoinable MPI. In Proceedings of the 10th IEEE International Workshop on Parallel and
Distributed Scientific and Engineering, PDSEC’09. IEEE.