Automatic Differentiation of Navier-Stokes Solvers

Thomas Kaminski, Ralf Giering, and Michael Voßbeck
and many collaborators

***Fast*Opt**

http://www.FastOpt.com

TU Berlin, January 2005

Outline

- FastOpt
- Automatic Differentiation
- TAF
- Applications
- TAC++
- Conclusions
- Demonstration

A few facts about *Fast*Opt

- Founded in February 2000 at Hamburg
- By Ralf Giering and Thomas Kaminski
- One more colleague as of July 2003:
  Michael Vossbeck (from TU Berlin)
- Two kinds of business:
  - Develop and provide tools (TAF)
    for Automatic Differentiation (AD)
  - Carry out Consulting Projects with focus on
    AD, Inverse Modelling, Optimisation...
- 16 years of Experience in AD

Intro to AD

Example:

$$F \quad : R^5 \rightarrow R^1$$
$$\quad : x \rightarrow y$$

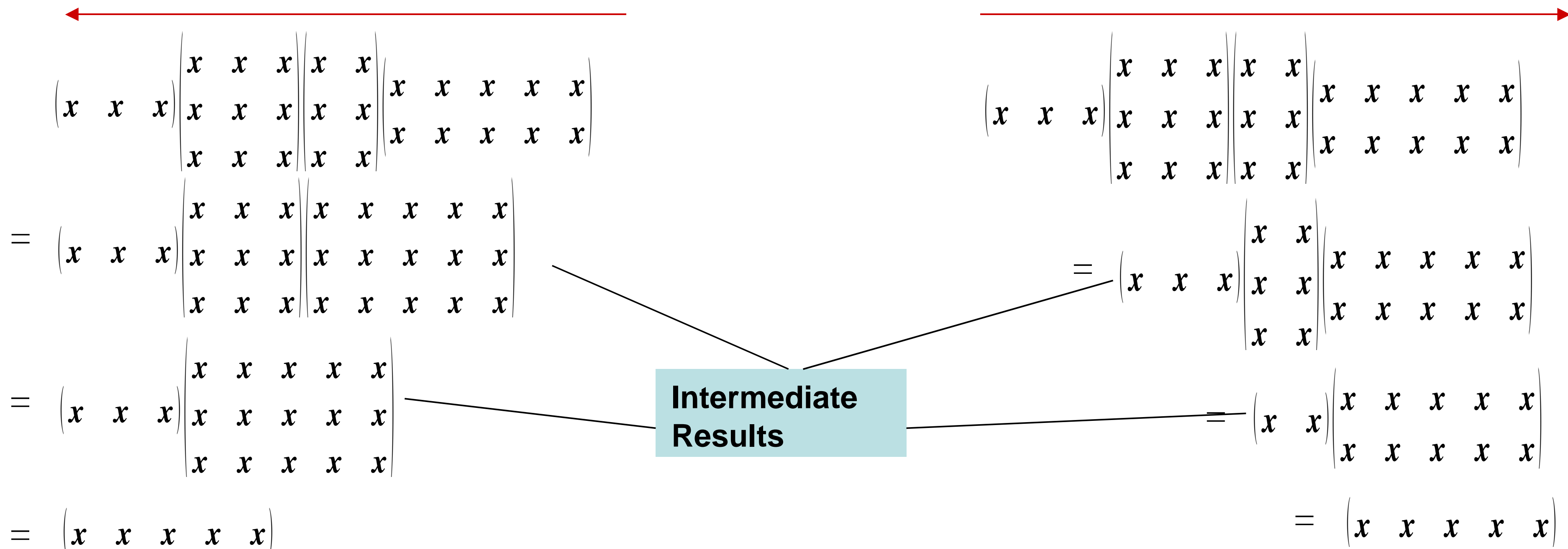
$$F(x) = f_4 \circ f_3 \circ f_2 \circ f_1(x),$$ **let each f_i be differentiable**

Apply the chain rule for Differentiation!

$$DF = Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1$$

FastOpt

AD: Forward vs. Reverse



- Forward and reverse mode yield the same result.
- Reverse mode: fewer operations (CPU time) and less space for intermediate results (memory)
- Cost for forward mode grows with N
- Cost for reverse mode grows with M

Reverse and Adjoint

$$DF = \overrightarrow{Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1}$$

$$DF^T = \overleftarrow{Df_1^T \cdot Df_2^T \cdot Df_3^T \cdot Df_4^T}$$

Propagation of Derivatives

Function:

$$x = z_0 \xrightarrow{f_1} z_1 \xrightarrow{f_2} z_2 \xrightarrow{f_3} z_3 \xrightarrow{f_4} z_4 = y$$

Forward:

$$x' = z'_0 \xrightarrow{Df_1} z'_1 \xrightarrow{Df_2} z'_2 \xrightarrow{Df_3} z'_3 \xrightarrow{Df_4} z'_4 = y'$$

$z'_0 \ldots z'_4$ are called tangent linear variables

Reverse:

$$\bar{x} = \bar{z}_0 \xleftarrow{Df_1^T} \bar{z}_1 \xleftarrow{Df_2^T} \bar{z}_2 \xleftarrow{Df_3^T} \bar{z}_3 \xleftarrow{Df_4^T} \bar{z}_4 = \bar{y}$$

$\bar{z}_0 \ldots \bar{z}_4$ are called adjoint variables

Forward Mode

Interpretation of tangent linear variables

Function:

$$x = z_0 \xrightarrow{f_1} z_1 \xrightarrow{f_2} z_2 \xrightarrow{f_3} z_3 \xrightarrow{f_4} z_4 = y$$

Forward:

$$x' = z'_0 \xrightarrow{Df_1} z'_1 \xrightarrow{Df_2} z'_2 \xrightarrow{Df_3} z'_3 \xrightarrow{Df_4} z'_4 = y'$$

x'=Id: $z'_2 = Df_2 \cdot Df_1 \cdot x' = Df_2 \cdot Df_1$

tangent linear variable z'_2 holds derivative of z_2 w.r.t. **x**: dz_2/dx

x'=v: $z'_2 = Df_2 \cdot Df_1 \cdot v$

tangent linear variable z'_2 holds directional derivative

of z_2 w.r.t. **x** in direction of **v**

*Fast*Opt

Function y=F(x) defined by Fortran code:

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y = v * w
```

Task: Evaluate **DF = dy/dx** in forward mode!

Problem: Identify f_1 f_2 f_3 f_4 z_1 z_2 z_3

Observation: f_3: `w = sin(u)` can't work, dimensions don't match!

Instead:

Just take all variables

$$f_3 : z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ w \\ y \end{pmatrix} \longrightarrow z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y \end{pmatrix}$$

A step in forward mode

$$f_3 \quad : \quad z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ w \\ y \end{pmatrix} \quad \rightarrow \quad z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y \end{pmatrix}$$

```
w        = sin(u)
```

$$z'_3 \quad = \quad Df_3 \quad z'_2$$

$$\begin{pmatrix} x'(1) \\ x'(2) \\ x'(3) \\ u' \\ v' \\ w' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \cos(u) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x'(1) \\ x'(2) \\ x'(3) \\ u' \\ v' \\ w' \\ y' \end{pmatrix}$$

```
gx(1) = gx(1)
gx(2) = gx(2)
gx(3) = gx(3)
gu    = gu
gv    = gv
gw    = gu*cos(u)
gy    = gy
```

Entire Function + required variables

Function code

```
u = 3*x(1)+2*x(2)+x(3)

v = cos(u)

w = sin(u)

y = v * w
```

Forward mode/ tangent linear code

```
gu = 3*gx(1)+2*gx(2)+gx(3)
 u = 3* x(1)+2* x(2)+ x(3)

gv = -gu*sin(u)
 v = cos(u)

gw = gu*cos(u)
 w = sin(u)

gy = gv*w + v*gw
 y = v * w
```

u v w are *required* variables, their values need to be provided to the derivative statements

Active and passive variables

Consider slight modification of code for y = F(x):

```
u   = 3*x(1)+2*x(2)+x(3)
pi  = 3.14
v   = pi*cos(u)
w   = pi*sin(u)
sum = v + u
y   = v * w
```

Observation: Variable `sum` (diagnostic) does not influce the function value y
Variable `pi` (constant) does not depend on the independent variables x

Variables that do influence y and are influenced by x are called *active variables*.
The remaining variables are called *passive variables*

Active and passive variables

Function code

```
u    = 3*x(1)+2*x(2)+x(3)

pi   = 3.14

v    = pi*cos(u)

w    = pi*sin(u)

sum = v + u

y    = v * w
```

Forward mode/ tangent linear code

```
gu = 3*gx(1)+2*gx(2)+gx(3)
u  = 3* x(1)+2* x(2)+ x(3)

pi = 3.14
gv = -gu*pi*sin(u)
v  = pi*cos(u)
gw = gu*pi*cos(u)
w  = pi*sin(u)


gy = gv*w + v*gw
```

For passive variables

- **no tangent linear variables needed**
- **no tangent linear statements for their assignments needed**

Reverse Mode

Function:

$$x = z_0 \xrightarrow{f_1} z_1 \xrightarrow{f_2} z_2 \xrightarrow{f_3} z_3 \xrightarrow{f_4} z_4 = y$$

Reverse:

$$\bar{x} = \bar{z}_0 \xleftarrow{Df_1^T} \bar{z}_1 \xleftarrow{Df_2^T} \bar{z}_2 \xleftarrow{Df_3^T} \bar{z}_3 \xleftarrow{Df_4^T} \bar{z}_4 = \bar{y}$$

$\bar{y}$=Id: $\bar{z}_2 = Df_3^T \cdot Df_4^T \cdot \bar{y} = (Df_4 \cdot Df_3)^T$

Adjoint variable $\bar{z}_2$ holds (transposed) derivative of **y** w.r.t. z_2: $d y / d z_2$

For example: y scalar, i.e. $\bar{y}=1$

A step in reverse mode

$$f_3 \quad : \quad z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ w \\ y \end{pmatrix} \longrightarrow z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y \end{pmatrix}$$

```
w = sin(u)
```

$$Df_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \cos(u) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\overline{z_2} = Df_3^T \, \overline{z_3}$$

$$\begin{pmatrix} \overline{x}(1) \\ \overline{x}(2) \\ \overline{x}(3) \\ \overline{u} \\ \overline{v} \\ \overline{w} \\ \overline{y} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cos(u) & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \overline{x}(1) \\ \overline{x}(2) \\ \overline{x}(3) \\ \overline{u} \\ \overline{v} \\ \overline{w} \\ \overline{y} \end{pmatrix}$$

```
adx(1) = adx(1)
adx(2) = adx(2)
adx(3) = adx(3)
adu = adu+adw*cos(u)
adv = adv
adw = 0.
ady = ady
```

FastOpt

Function code

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y = v * w
```

Adjoint code

```
   u = 3*x(1)+2*x(2)+x(3)
   v = cos(u)
   w = sin(u)

adv = adv+ady*w
adw = adw+ady*v
ady = 0.

adu = adu+adw*cos(u)
adw = 0.

adu = adu-adv*sin(u)
adv = 0.

adx(3) = adx(3)+3*adu
adx(2) = adx(2)+2*adu
adx(1) = adx(1)+adu
adu    = 0.
```

Function F defined by Fortran code:

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y = v * w
```

Typically, to save memory, variables are used more than once!

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
u = sin(u)
y = v * u
```

Function code

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
u = sin(u)
y = v * u
```

Adjoint code

```
   u = 3*x(1)+2*x(2)+x(3)
   v = cos(u)
   u = sin(u)
adv = adv+ady*u
adu = adu+ady*v
ady = 0.
   u = 3*x(1)+2*x(2)+x(3)

adu = adu*cos(u)

adu = adu-adv*sin(u)
adv = 0.

adx(3) = adx(3)+3*adu
adx(2) = adx(2)+2*adu
adx(1) = adx(1)+adu
adu    = 0.
```

Store and retrieve values

Function code

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
u = sin(u)
y = v * u
```

Adjoint code

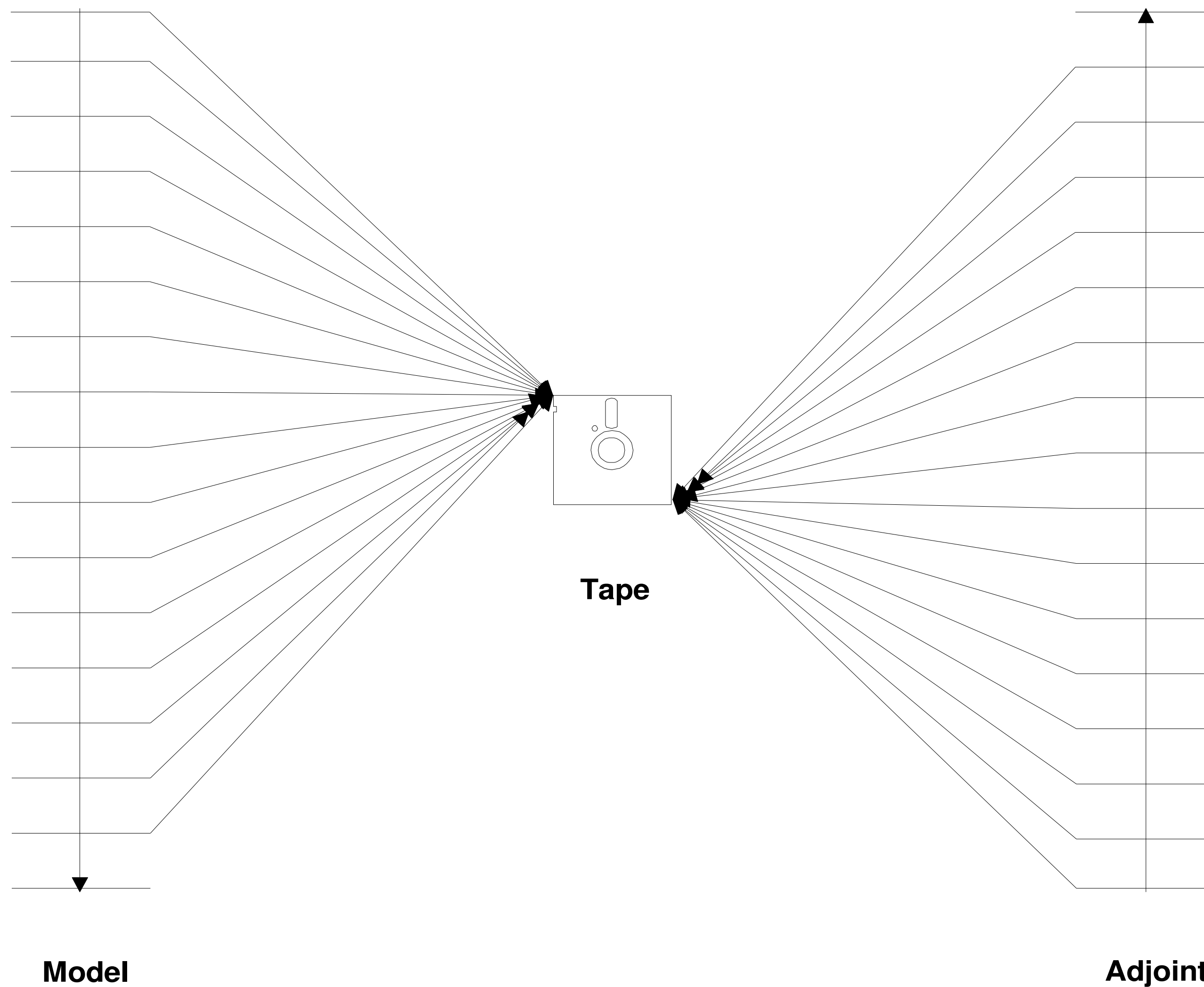
```
u = 3*x(1)+2*x(2)+x(3)
   store (u)
v = cos(u)
u = sin(u)
adv = adv+ady*u
adu = adu+ady*v
ady = 0.
   retrieve (u)
adu = adu*cos(u)
adu = adu-adv*sin(u)
adv = 0.
adx(3) = adx(3)+3*adu
adx(2) = adx(2)+2*adu
adx(1) = adx(1)+adu
adu    = 0.
```

Bookkeeping must be arranged
(store / retrieve)

Values can be saved

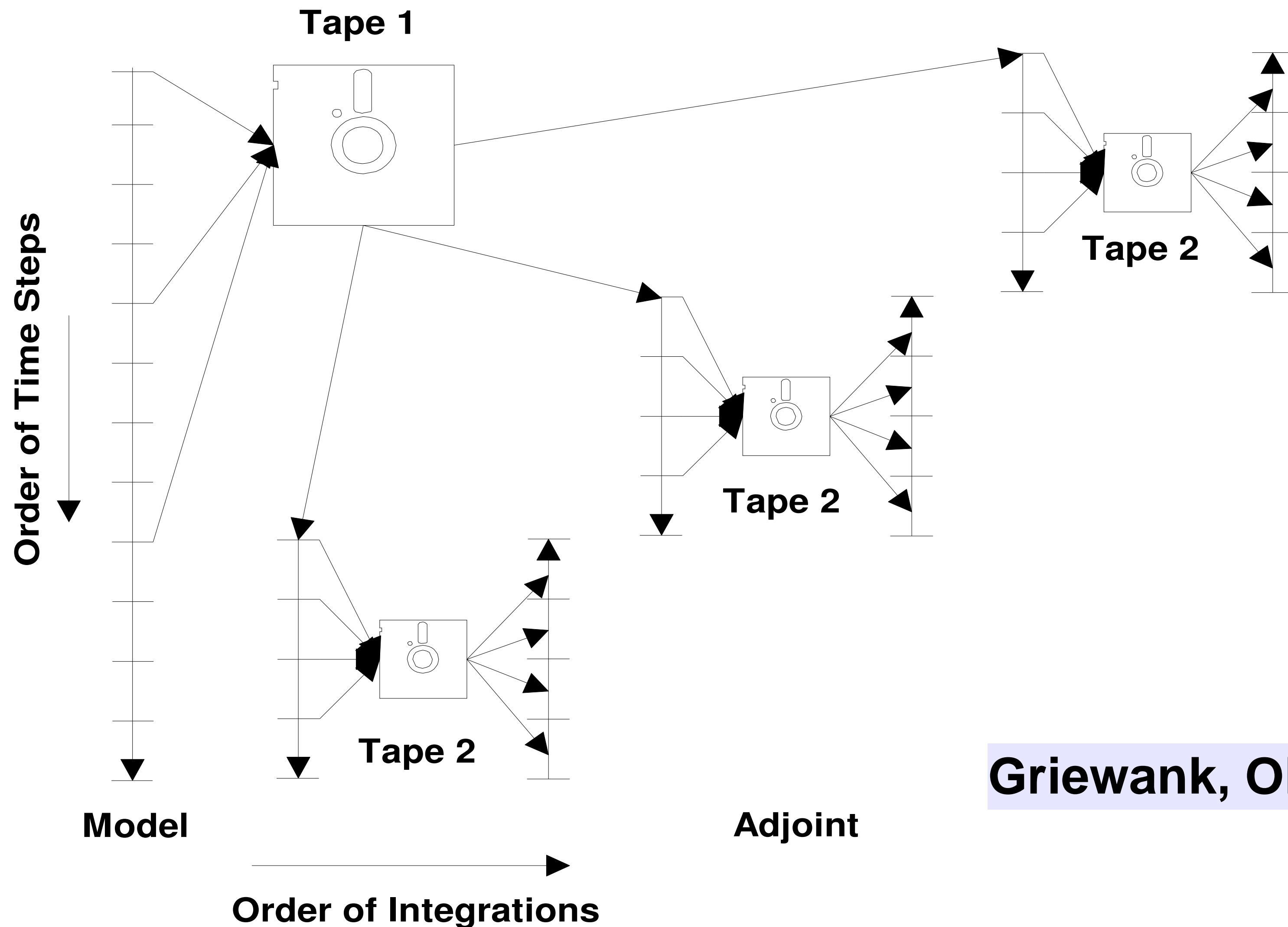
- on disc or
- in memory

Storing of required variables



FastOpt

Checkpointing



Griewank, OMS, 1992

FastOpt

Vector valued function in Reverse Mode

$F \; : R^5 \rightarrow R^2$

$\; : x \rightarrow y$

$$DF = Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1$$

$$\begin{pmatrix} x & x & x \\ x & x & x \end{pmatrix}\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}\begin{pmatrix} x & x \\ x & x \\ x & x \end{pmatrix}\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x & x \\ x & x & x \end{pmatrix}\begin{pmatrix} x & x \\ x & x \\ x & x \end{pmatrix}\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x \\ x & x \end{pmatrix}\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

Intermediate Results (adjoint Variables)

z_1, z_2, ...

get one more dimension

Adjoint code for vector valued function

Function code

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y(1) = v * w
y(2) = v + w
```

Adjoint code

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)

do i=1,2
     adv(i) = ady(i,2)
     adw(i) = ady(i,2)
     ady(i,2) = 0.
enddo

do i=1,2
     adv(i) = adv(i)+ady(i,1)*w
     adw(i) = adw(i)+ady(i,1)*v
     ady(i,1) = 0.
enddo

do i=1,2
     adu(i) = adu(i)+adw(i)*cos(u)
     adw(i) = 0.
enddo

do i=1,2
     adu(i) = adu(i)-adv(i)*sin(u)
     adv(i) = 0.
enddo

and so on...
```

AD Summary

- **AD exploits chain rule**
- **Forward and reverse modes**
- **Active/Passive variables**
- **Required variables: Recomputation vs. Store/Reading**
- **Scalar and vector modes**

Further Reading

- AD Book of **Andreas Griewank**: *Evaluating Derivatives: Principles of Algorithmic Differentiation, SIAM, 2000*
- Books on **AD Workshops:**
  Nice 2000: *Corliss et al. (Eds.), Springer*
  Santa Fe 1996: *Berz et al. (Eds.), SIAM*
  Beckenridge 1991: *Griewank and Corliss (Eds.), SIAM*
- **Olivier Talagrand's** overview article in Santa Fe Book
- RG/TK article: *Recipes of Adjoint Code Construction, TOMS, 1998*

AD Tools

- Specific to programming language
- Source-to-source / Operator overloading
- For details check http://www.autodiff.org !

Selected Fortran tools (source to source):

- ADIFOR (M. Fagan, Rice, Houston)
- Odyssee (C. Faure) -> TAPENADE (L. Hascoet, INRIA, Sophia-Antipolis, France)
- TAMC (R. Giering) -> TAF (FastOpt)

Selected C/C++ tools:

- ADOLC (A. Walther, TU-Dresden)
- ADIC (P. Hovland, Argonne, Chicago)

TAF
Transformation of Algorithms in Fortran

- Source-to-source translator for Fortran-77/95
- Commercial successor of TAMC
- Forward and reverse mode (1st derivatives):
  Tangent linear and adjoint models
- Scalar and vector mode
- Efficient Hessian (2nd derivative) code
  by applying TAF twice (e.g. forward over reverse)
- Command line program with many options
- TAF-Directives are Fortran comments
- Extensive and complex code analyses
  (similar to optimising compilers)
- Generated code is structured and well readable

TAF
More features/improvements

- Generation of flexible store/read scheme for required values triggered by TAF init and store directives
- Generation of simple checkpointing scheme (Griewank, 1992) triggered by combination of TAF init and store directives
- Generation of efficient adjoint (Christianson, 1996, 1998) for converging iterations triggered by TAF loop directive
- TAF flow directives for black-box routines, or to include user provided derivative code (exploit self-adjointness, MPI wrappers, etc...)
- Automatic Sparsity Detection
- Basic support for MPI and OpenMP

- TAF is constantly being adapted to new language standards
- TAF code analyses are constantly being extended
- TAF algorithms are constantly being improved and adapted to user needs

Variational Data Assimilation

Notation:

$\underline{\mathbf{s}}$: state vector

t : time

$\underline{\mathbf{d}}$: vector of observations

$\underline{\boldsymbol{\sigma}}$: vector observational uncertainties

Principle:

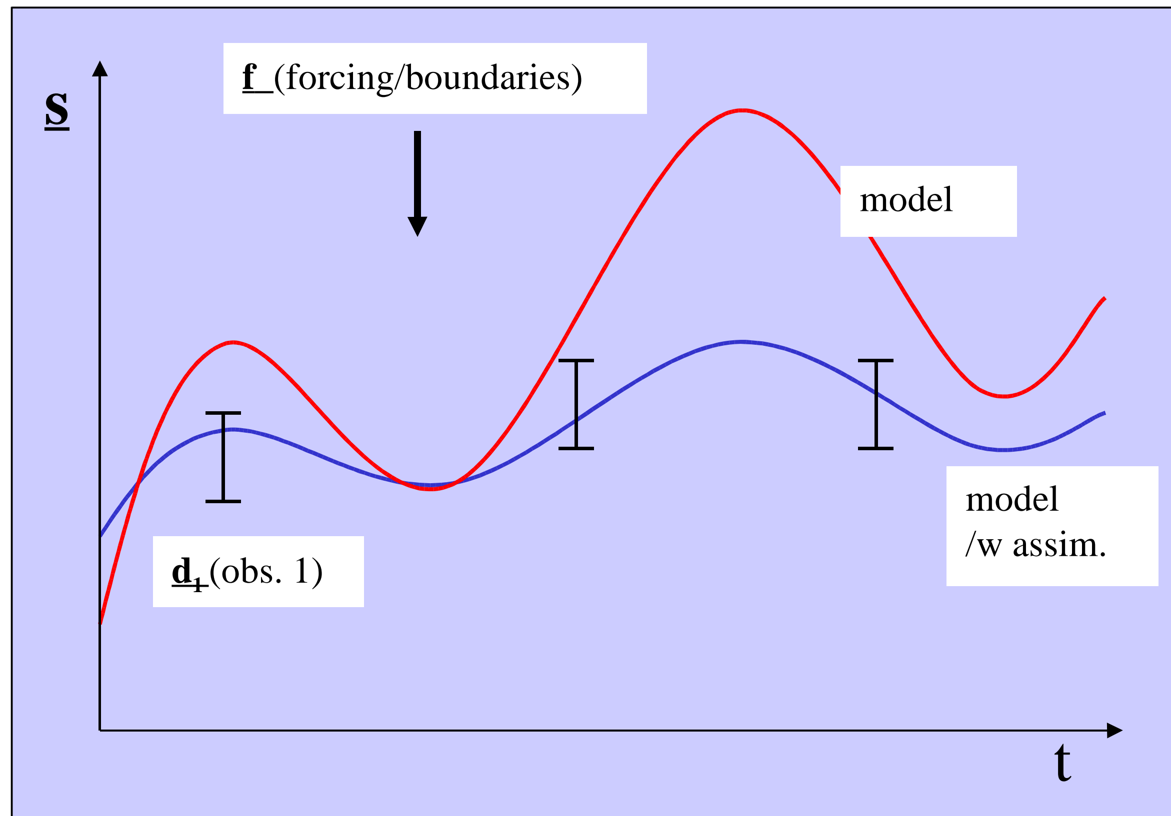
- define vector of control variables $\underline{\mathbf{x}}$, e.g.,
  - initial state ($\underline{\mathbf{s}}_0$)
  - forcing/boundary conditions ($\underline{\mathbf{f}}$)
  - internal model parameters ($\underline{\mathbf{p}}$)
- define quality of fit by cost function:

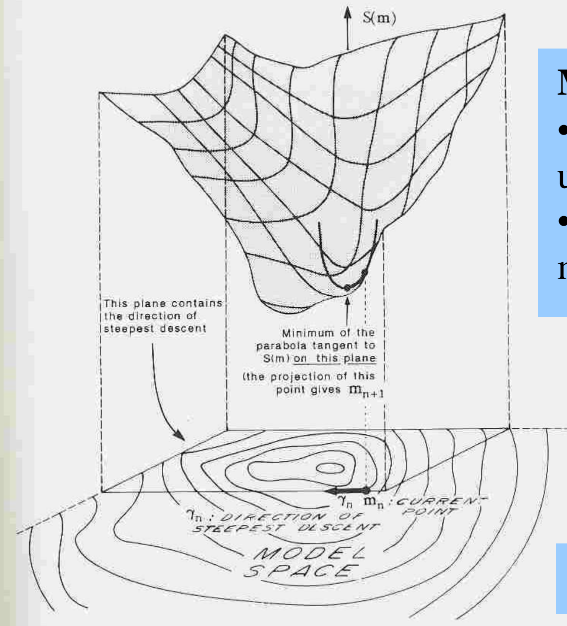
$$J(\mathbf{x}) = 1/2 \ \Sigma \ (\underline{\mathbf{d}}_i - \underline{\mathbf{s}}_i(\mathbf{x}))^2 / \ \underline{\boldsymbol{\sigma}}_i^2$$

- minimise J($\mathbf{x}$) by variation of $\mathbf{x}$

Remarks:

- can handle ***any*** observation that can be computed from the model state
- in numerical weather prediction (NWP) 4DVar usually variation of initial state ($\underline{\mathbf{s}}_0$) only
- can also include uncertainties from model error and correlated uncertainties
- can include more constraints in J, next slide...





Minimisation:

- efficient minimisation algorithms use gradient of J
- gradient provided by adjoint model

Figure: *Tarantola* (1997)

MIT global ocean model
http://mitgcm.org

Slides: Courtesy
Patrick Heimbach, MIT
(details: Heimbach et al., LNCS. 2003)

ECCO state estimation: problem size

- Dimensionality:
  - grid @ $1° \times 1°$ resolution: $n_x \cdot n_y \cdot n_z = 360 \cdot 160 \cdot 23$ — $1,324,800$
  - model state: 17 3D + 2 2D fields — $\sim 2 \cdot 10^7$
  - timesteps: 10 years @ 1-hour time step — $87,600$
  - control vector — $\sim 1 \cdot 10^8$
    - initial temperature (T), salinity (S)
    - time-dependent surface forcing (every 2 days)
  - cost function: observational elements: — $\sim 1 \cdot 10^8$
- Computational size:
  - 60 processors (15 nodes) @ 512MB per proc.
  - I/O: 10 GB input, 35GB output
  - time: 59 hours per iteration @ 60 processors
- What we would ideally want:
  - $1/10° \times 1/10°$ resol., 1000 years, full model error covariance ...

The ECCO costfunction – observational elements

$\mathcal{J} = (\bar{\eta} - \bar{\eta}_{TP})^t \mathbf{W}_{\mathbf{geoid}} (\bar{\eta} - \bar{\eta}_{TP})$	TOPEX absolute SSH
$+ (\eta - \eta'_{TP})^t \mathbf{W}_{\mathbf{TP}} (\eta - \eta'_{TP})$	TOPEX SSH anomalies
$+ (\eta - \eta'_{ERS})^t \mathbf{W}_{\mathbf{ERS}} (\eta - \eta'_{ERS})$	ERS SSH anomalies
$+ (\bar{T}_{surf} - \bar{T}_{Reyn})^t \mathbf{W}_{\mathbf{SST}} (\bar{T}_{surf} - \bar{T}_{Reyn})$	Reynolds SST
$+ (\bar{T} - \bar{T}_{Lev})^t \mathbf{W}_{\mathbf{T_{Lev}}} (\bar{T} - \bar{T}_{Lev})$	Levitus clim.
$+ (\bar{S} - \bar{S}_{Lev})^t \mathbf{W}_{\mathbf{S_{Lev}}} (\bar{S} - \bar{S}_{Lev})$	Levitus clim.
$+ (\tau_x - \tau_{x,NCEP})^t \mathbf{W}_{\tau_{\mathbf{x}}} (\tau_x - \tau_{x,NCEP})$	zonal wind stress
$+ (\tau_y - \tau_{y,NCEP})^t \mathbf{W}_{\tau_{\mathbf{y}}} (\tau_y - \tau_{y,NCEP})$	merid. wind stress
$+ (H_Q - H_{Q,NCEP})^t \mathbf{W}_{\mathbf{H_Q}} (H_Q - H_{Q,NCEP})$	NCEP heat flux
$+ (H_F - H_{F,NCEP})^t \mathbf{W}_{\mathbf{H_F}} (H_F - H_{F,NCEP})$	NCEP freshwater flux

Currently added:

- Jason-1 altimetry (sea surface height)
- WOCE hydrography, XBT, TAO buoys
- PALACE/ARGO tracer profiles and drift velocities
- surface drifter velocities
- NSCAT/QuickScat surface wind stress fields
- TRMM/TMI tropical surface temperature fields

Slides: Courtesy
Patrick Heimbach, MIT
(details: Heimbach et al., LNCS. 2003)

- **Fast and reliable gradient (10^8 components)**
- **Scientific applications infeasible without efficient adjoint code**
- **After each update of MITgcm, the derivative code is updated by TAF (automated process)**

AD of finite volume GCM

Collaboration with NASA/GMAO (Todling, Errico, Gelaro,Winslow)

- finite volume GCM (fvGCM) is the global weather forecast model of NASA/GMAO
- consistes of dynamical core (Navier-Stokes solver) + physics (parametrisations for clouds etc)
- dynamical core by Lin and Rood (1996, 1997) and Lin (1997)
- state comprises two horizontal wind components, pressure difference, potential temperature, and moisture
- various spatial resolutions and integration periods
- GMAO's data assimilation system requires tangent linear (TLM) and adjoint (ADM) codes of fvGCM's dynamical core
- Further applications include singular vector analysis, adjoint sensitivity studies, and inverse modelling of tracers

finite volume GCM

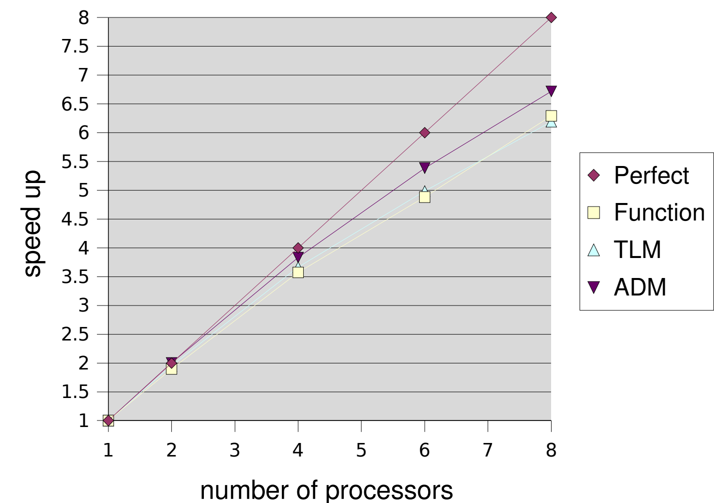
- for most applications, TLM and ADM need to linearise around an external state-trajectory provided by an integration at higher spatial resolution including full physics
- dynamical core comprises 87'000 lines of Fortran 90 code, excluding comments
- uses features such as
  free source form, derived types, allocatable arrays
- parallelises (domain decomposition) using *OpenMP, MPI*, or both
- good performance TLM and ADM crucial for applications

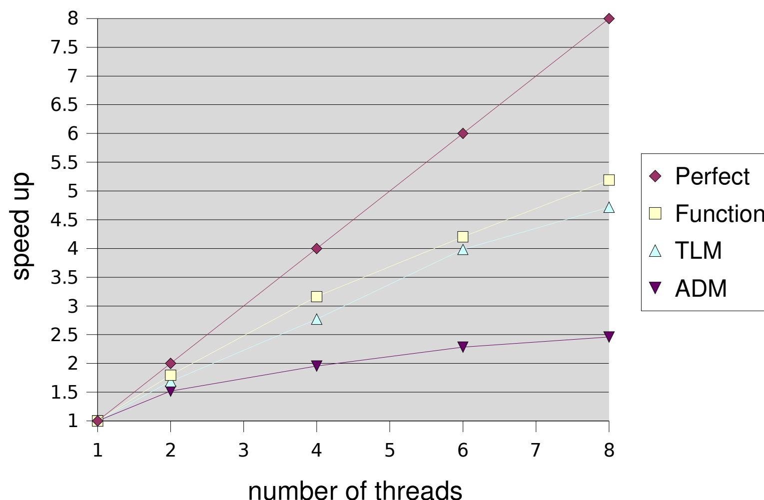
- TLM and ADM generation by fully automated procedure via TAF

AD of fvGCM

Parallelisation: MPI/OpenMP

- Model has wrapper routines (e.g. mp_send3d_ns) that call the respective MPI library routines (e.g. mpi_isend)
- Wrappers are encapsulated in one module
- Decision between MPI-1/2 happens in wrappers
- In forward mode, TAF handles (most) MPI calls.
  We need, however, TLM and ADM
  -> Construction of MPI in TLM and ADM at level of wrappers
  - Inserting of TAF flow directives for wrappers
  - TLM and ADM wrapper routines hand written
  - TLM and ADM wrappers reuse model wrappers (easy to maintain)
  - Handling of MPI-1 and MPI-2 at once
- Encapsulation helped a lot!





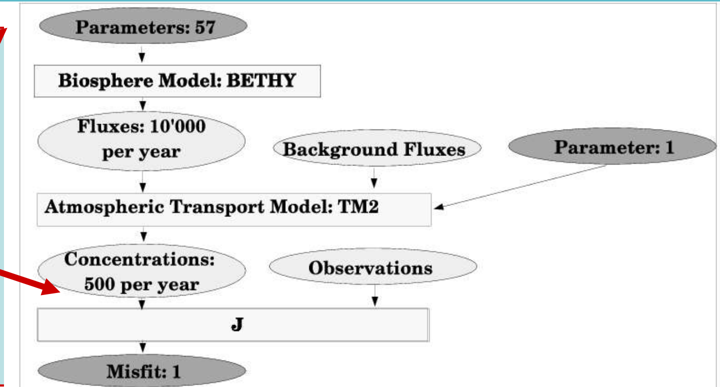
- Model uses only a single directive:
  !$omp parallel do
- TAF analyses the loop-carried dependencies
- For ADM loop, according to the dependencies, TAF generates the proper !$omp directive for the adjoint loop and (if necessary) additional statements to preserve parallelism
- Can generate code for OpenMP-1 or OpenMP-2
- OpenMP-1 adjoint of fvGCM need many critical sections, because OpenMP-1 does not support array reductions.
- OpenMP-2 does and thus yields faster code.
- For TLM loop, TAF uses the similar directive

Terrestrial Vegetation Modelling (http://CCDAS.org)

BETHY coupled to TM2

Collaboration with Knorr, Scholze, Widmann (Max-Planck, Germany) and Rayner (CSIRO, Australia)

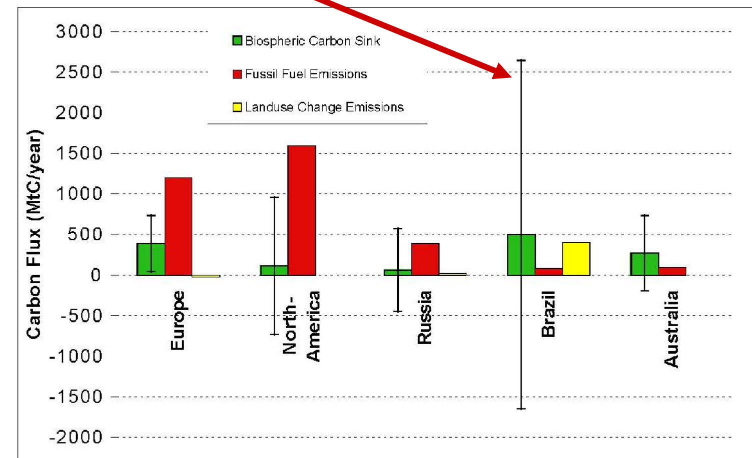
- Terrestrial vegetation model (BETHY, Knorr 2000) simulates CO2 fluxes to atmosphere
- Model has 58 unkown parameters m ("design variables")
- Atmospheric CO2 is observed at global networks
- Use atmospheric CO2 to constrain parameters (calibration)
- Put mitfit to observations and other information in scalar objective function J
- Use gradient algorithm to minimise objective function-> Need adjoint
- Use error in measurements (and model)
  to infer uncertainties in parameters (covariance C_m) -> Need Hessian
- Use parameter uncertainties to derive uncertainties for predictions -> Need Jacobian



$$C_m \approx \left\{ \frac{\partial^2 J(m_{opt})}{\partial m_{i,j}^2} \right\}^{-1}$$

$$C_y = \left(\frac{\partial y_i(m_{opt})}{\partial m_j} \right) C_m \left(\frac{\partial y_i(m_{opt})}{\partial m_j} \right)^T$$

BETHY:
5400 lines of Fortran 90 code, e.g. Modules, dynamic memory allocation, derived types
intrinsics like reshape, matmul, transpose



AD of BETHY with TAF by FastOpt:
- Generated adjoint, Jacobian, and Hessian, and sparsity detection code
- Inserting of 38 TAF directives for storing, support of code analysis

- First system to optimise arbitrary number of vegetation parameters and with rigorous handling of uncertainty
- Model is continuously developed further, TAF is integral part of modelling system, updates derivative code

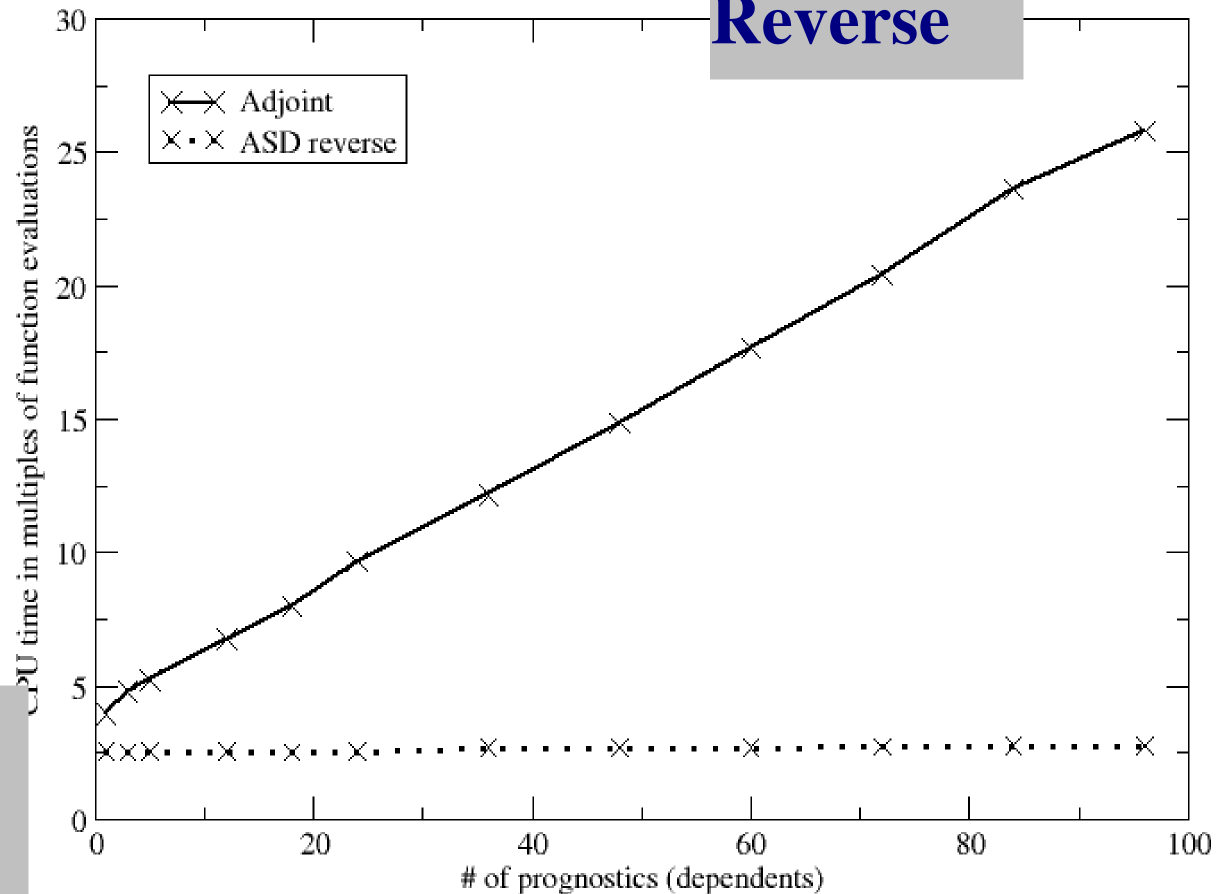
Jacobian and ASD Performance

Forward

58 independents

Jacobian: 12

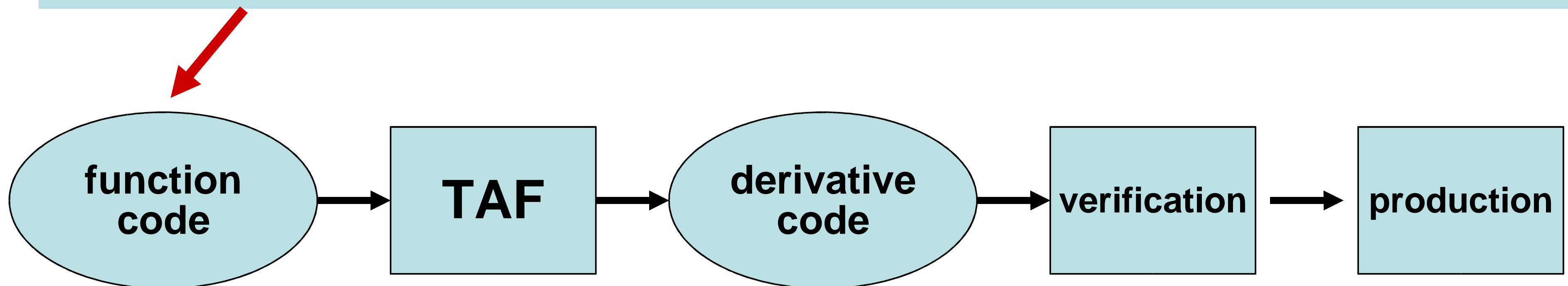
ASD: 1.3



CPU time quantified in multiples of function evaluations

Applications: FastOpt strategy

- Apply combination of modifying function code and TAF enhancements
- For a code under development, the major effort is to make it TAF compatible once, keeping the updates TAF compatible is usually easy.
- Generated derivative code must always be verified before being used in production
- Optimise generation of derivative code, don't modify generated code by hand
  This is essential to automate the process chain from
  new release of function code to production



Aerodynamics: NSC2KE

with T. Slawig, TU-Berlin

- Model: NSC2KE by Bijan Mohammadi (1994)
- Mixed finite volume-finite element Galerkin CFD model
- 2 dimensional on unstructured grid
- Euler Part with Roe-, Osher-, or Kinematic solver
- K-epsilon turbulence model
- 4th order Runge Kutta
- 2500 lines of Fortran 77 code without comments
- Previous AD applications of this code by:
  - Mohammadi et al. (1994) w/ Odyssée (Rostaing et al, 1993)
  - Hovland et al. (1997), Slawig (2001) w/ ADIFOR (Bischof et al. '96)
  - S. Ulbrich (2001) w/ TAMC (Giering, 1997)

Aerodynamics

Automatic Differentiation of NSC2KE

AD for simple Euler Configuration,
derivative of lift/drag w.r.t. angle of attack and mach number
Adjoint:

- 16 store directives inserted
- Used TAF iteration directive to trigger efficient write/read scheme (Christianson, 1996, 1998):
  stores only steady state values of required variables
- Required variables kept in memory: 1.5 MB
- CPU(gradient+function)/CPU(function) = 3.4

Tangent linear:

- CPU(derivative+function)/CPU(function) = 2.4

Hessian:

- Forward over Reverse mode
- CPU(Hessian*1 vector)/CPU(function) = 9.8

Test with one customer:
Design sensitivities for 68 coordinates of wing contours:

Adjoint not optimised for speed:
CPU(gradient+function)/CPU(function) ≈ 6

Directive to initialise tape for storing

Iteration directive

Directives to arrange storing

```
c$taf init tape1 = static, 1
c$taf loop = iteration, ua, un, pres
      do ktout = 1, ktmax
         kt = kt0 + ktout
c$taf store pres,reyturb,ua,t = tape1
            call caldtl(dtmin,dt)
c$taf store reylam,dtl          = tape1
            call runge( kt0, som )
      end do
```

some larger TAF Derivatives

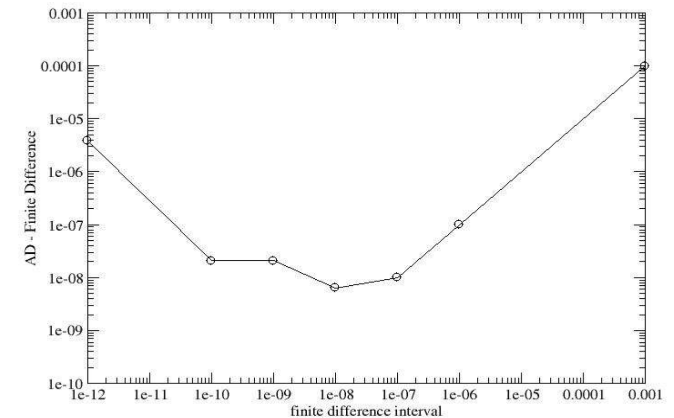
Model (Who)	Lines	Lang	TLM	ADM	Ckp	HES
NASA/GMAO (w. Todling, Errico, Gelaro	87'000	F90	1.5	3.9—11	-	-
MOM3 (Galanti & Tziperman)	50'000	F77	Yes	4.6	2 lev	-
MITGCM (ECCO Consortium)	100'000	F77	1.8	5.5	3 lev	11.0/1
BETHY (w. Knorr, Rayner, Scholze)	5'400	F90	1.5	3.6	2 lev	12.5/5
Nav.-Stokes-Solver (Hinze, Slawig)	450	F77	-	2.0	steady	-
NSC2KE (w. Slawig)	2'500	F77	2.4	3.4	steady	9.8/1
HB_AIRFOIL (Thomas & Hall)	8'000	F90	-	3.0		-

- Lines: total number of Fortran lines without comments
- Numbers for TLM and ADM give CPU time for (function + gradient) relative to forward model
- HES format: CPU time for Hessian * n vectors rel. t. forw. model/ n
- 2 (3) level checkpointing costs 1 (2) additional model run(s)

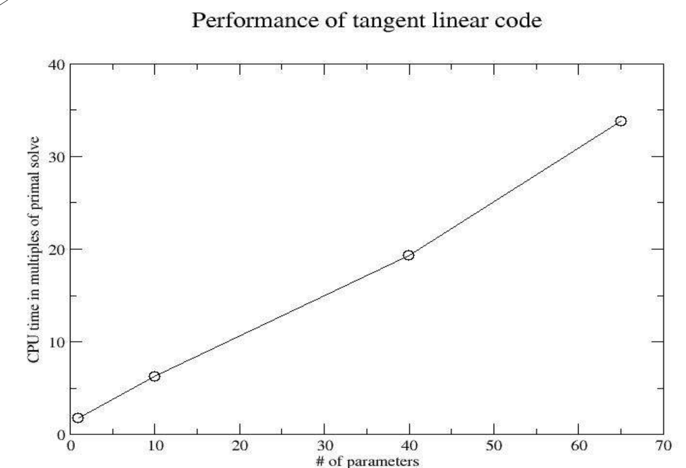
http://FastOpt.com/references/cfd.h

MEGADESIGN

- “Verbundvorhaben, Luftfahrtforschungsprogramm”:
  June 03 – April 07
- Co-ordinated by DLR, futher participants:
  - Airbus Deutschland, EADS-M
  - CLE, FastOpt, Synaps
  - RWTH, TU Berlin, TU Braunschweig, TU Darmstadt, Uni Trier



- One of many MEGADESIGN tasks:
  Adjoint of tubulence in DLR CFD code FLOWer
  joint work with DLR (B. Eisfeld, N. Gauger, N. Kroll, etc...)
- Tangent linear code verified for
  first very simple test configuration:
  2d, NACA12, Wilcox turbulence model, finest grid, two time steps
  AD of lift w.r.t. angle of attack
- Performance o.k.
- Adjoint in the pipe ...



Performance and accuracy

Instantaneous control of Navier-Stokes

with M. Hinze TU Dresden Opt. Meth. Software 18/3 (2003)

$$\min_u \hat{J}(u) := \frac{1}{2}\|y(u) - z\|^2_{L^2(Q)} + \frac{\gamma}{2}\|u\|^2_{L^2(Q)}$$

$$\begin{aligned} y_t - \nu\Delta y + y\cdot\nabla y &= u && \text{in } Q := \Omega\times[0,T] \\ y &= g && \text{on } \partial\Omega\times[0,T] \\ y(0) &= y_0 && \text{on } \Omega. \end{aligned}$$

Time discretization:

$$y^j \approx y(t_j)$$

Slides: Courtesy
Thomas Slawig, TU Berlin
(details: Hinze and Slawig, OMS, 2003)

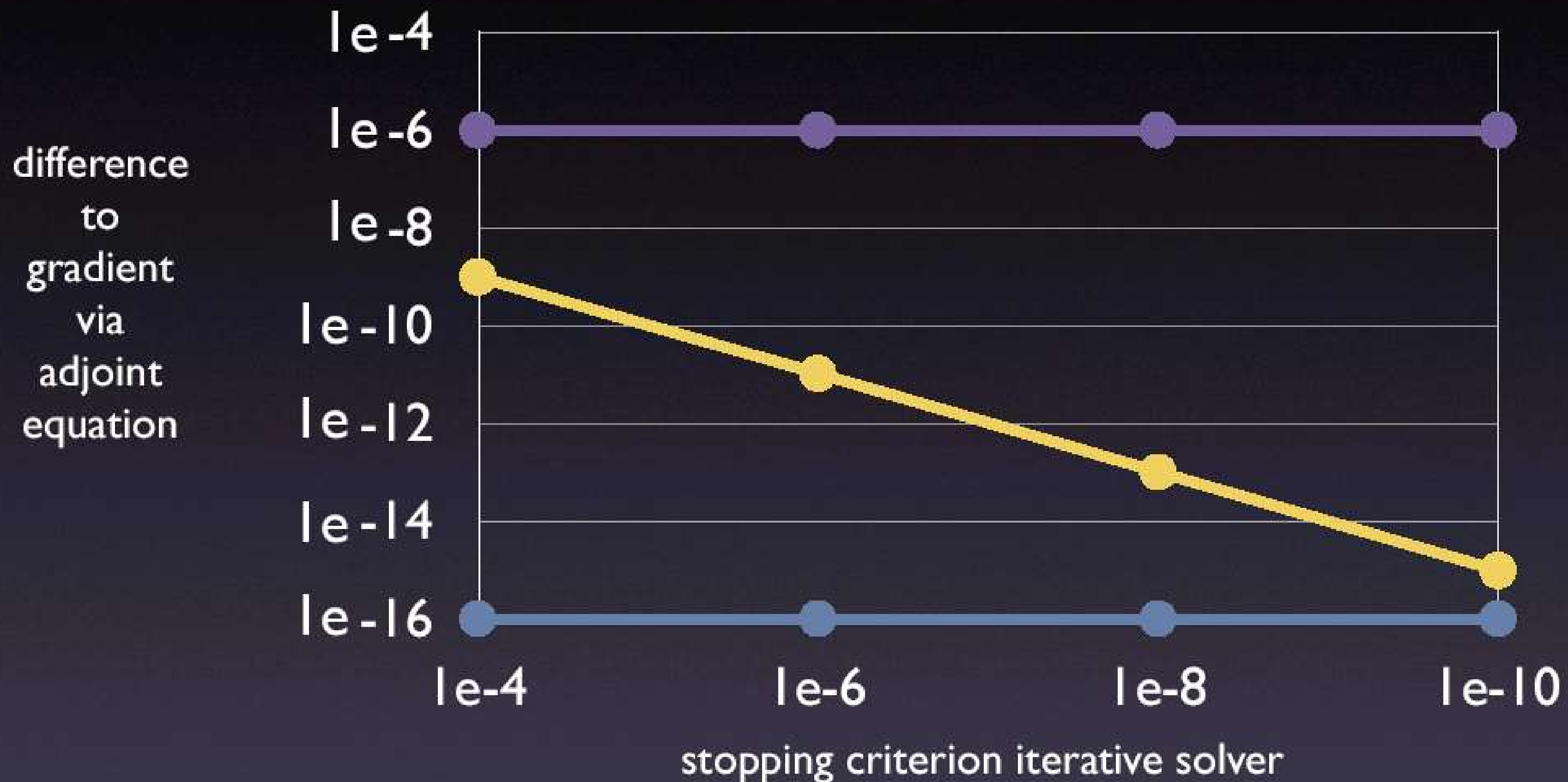
Minimize

$$J(u^j) = \frac{1}{2}\|y^j - z^j\|^2_{L^2(\Omega)} + \frac{\gamma}{2}\|u^j\|^2_{L^2(\Omega)}$$

s.t. time-discrete (elliptic) Navier-Stokes

Accuracy

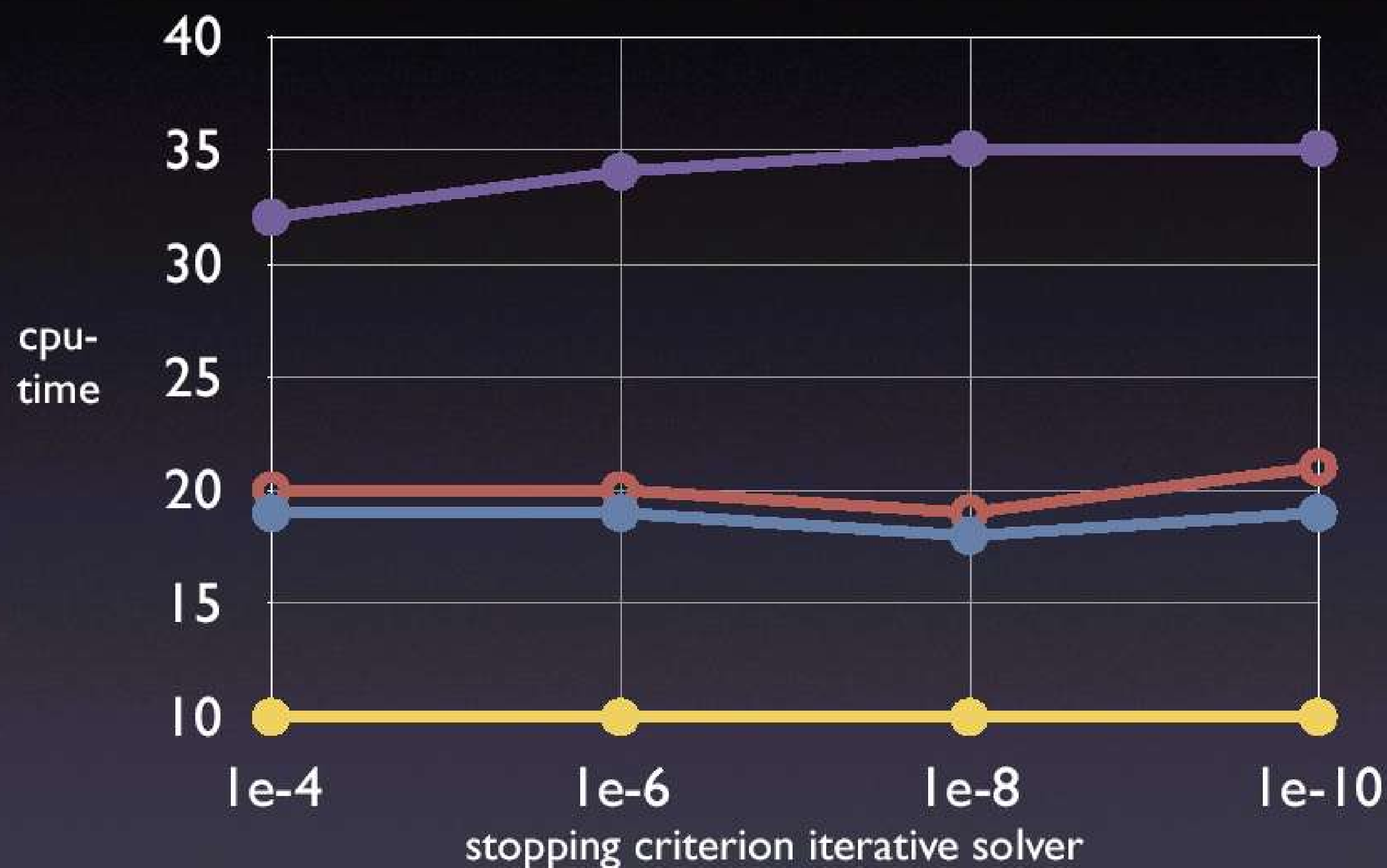
State equation solver includes iterative (cg) solvers



- AD (TAF) with ...
- ... without differentiation of iterative solver
- finite differences

Performance

relative w.r.t. function evaluation



function

discretized adjoint equation

AD (TAF) with ...

... without iterative solver

Performance ratios

TAF vs hand coded adjoints

Code	Hand	TAF/TAMC	relativ
EPT (MINPACK-2)	1.5	1.9	-27%
GL1 (MINPACK-2)	1.5	1.7	-13%
GL2 (MINPACK-2)	2.1	1.3	38%
MSA (MINPACK-2)	1.8	1.6	9%
PJB (MINPACK-2)	1.8	2.2	-26%
SSC (MINPACK-2)	1.2	1.1	4%
Nav.-Stokes (Hinze and Slawig)	1.9	2.0	-5%
EULOSOLDO (Cusdin and Müller)	2.4	2.7	-12%
3D NS CFD-code (Cusdin and Müller	1.2	1.2	-4%

=> Performance of TAF generated adjoints is comparable to that of hand written adjoints

*Fast*Opt

TAC++
Transformation of Algorithms in C++

- Source-to-source translator for C/C(++)
- Same approach as TAF
- Completed basic feasibility study (Vossbeck et al., submitted 2004)
- Generated adjoint of Roe solver
  - from EULSOLDO by Müller (1991)
  - 129 statements in C code (by f2c from Fortran-77)
  - C code contains simple constructs
  - Performance comparable to TAF generated code
- Functionality will be extended "model by model“
  (Parts of DLR C-code TAUIJ differentiated in collaboration with DLR)

Conclusions

- TAF generates efficient tangent linear, adjoint, and Hessian code
- TAF is an essential component in a number of modeling systems
- Number of CFD applications increasing
- First feasibility study for TAC++ completed

More info:
http://www.FastOpt.com