First applications of FastOpt's new AD tool TAF

Thomas Kaminski and Ralf Giering

*Fast*Opt

Web: http://www.FastOpt.de

5th Workshop on Adjoint Applications in Dynamic Meteorolgy, Pennsylvania, April 2002

A few facts about FastOpt

- Founded in February 2000 at Hamburg
- By Ralf Giering and Thomas Kaminski
- Two kinds of business:
  - Develop and provide tools (TAF) for Automatic Differentiation (AD)
  - Carry out Consulting Projects with focus on AD, Inverse Modelling, Optimisation...
- 13 years of Experience in AD

Overview

- Facts about FastOpt
- AD Tool TAF: Introduction
- AD Tool TAF: Work for the user
- Differentiated Models / Performance
- Future Developments

Fortran AD Tools
see www.autodiff.org for more info

- ADIFOR−2 (F77): TLM
  - Future version 3 will support ADM
- TAF (F77 to F95) –> ...
- TAMC (F77): TLM + ADM
  - remote access to server
  - no more maintainance
- TAPENADE (F77 to F95): TLM + ADM
  - successor of Odyssee
  - accessible through web or download

TAF: Transformation of Algorithms in Fortran

- Source−to−source translator for Fortran−77 to 95
- Commercial successor of TAMC
- Forward and reverse mode (1st derivatives):
  Tangent linear (TLM) and adjoint models (ADM)
- Efficient Hessian (2nd derivative) code
  by applying TAF twice (forward over reverse mode)
- 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: Examples of work for the user

- Prepare model for multiple runs
  - Avoid reading from sequential files (e.g. forcing)
  - Split initializations off the code to be differentiated
- Avoid complex control flow structures
- Keep allocation/deallocation structure simple (static)
- Optimize performance of ADM by
  - inserting loop directives
  - inserting flow directives
  - inserting store directives

Reading of sequential files: Bad example

```
c open file containing boundary conditions
      open (10, file='boundary.txt')

c time integration
      do itime = 1, ntime
        state_old = state_new
        read (10) boundary
        state_new = dostep (state_old, boundary)
      enddo

      close (10)
```

Problem: The proper boundary values are not accessible in reverse order

Reading of sequential files: Good example

Model Code:

```
c open file containing boundary conditions
      open (10, file='boundary.txt', ACCESS='DIRECT',RECL=...)
c time integration
      do itime = 1, ntime
         state_old = state_new
         read(10,rec=itime) boundary
         state_new = dostep (state_old, boundary)
      enddo
      close (10)
```

Corresponding Adjoint Code:

```
      open (10, file='boundary.txt', ACCESS='DIRECT',RECL=...)
      do itime = ntime, 1
         read(10,rec=itime) boundary
         call ad_dostep (adstate_new, state_old, adstate_old, boundary)
         adstate_new = 0.
         adstate_new = adstate_new + adstate_old
         adstate_old = 0.
      enddo
      close (10)
```

Complex Control Flow 1: If then else expressed with gotos

```
C=============================================
C     compute sqrt(x), but avoid x = 0
C=============================================
      SUBROUTINE MODEL( N, X, Y )
      implicit none
      INTEGER N
      REAL    X(N), Y

      real epsilon
      parameter (epsilon=0.0001)

      if ( abs(x(1)) .lt. epsilon) goto 100
      y = sqrt (x(1))
      goto 200

 100  continue
      print*,' Attention: x close to zero (sqrt not differentiable)'
      y = 0.

 200  continue

      END
```

Complex Control Flow 1: Normalised Structure and Adjoint

```
      subroutine admodel( n, x, adx, y, ady )
C*********************************************************************
C** This routine was generated by Automatic differentiation.      **
C** FastOpt: Transformation of Algorithms in Fortran, TAF 1.3.14 **
C*********************************************************************
... ( thomas:initialization removed by hand -> fit on page) ...
C-----------------------------------------
C FUNCTION AND TAPE COMPUTATIONS
C-----------------------------------------
      if (abs(x(1)) .lt. epsilon) then
        y = 0.
      else
        y = sqrt(x(1))
      endif
C-----------------------------------------
C ADJOINT COMPUTATIONS
C-----------------------------------------
      if (abs(x(1)) .lt. epsilon) then
      else
        adx(1) = adx(1)+ady*(1./(2.*sqrt(x(1))))
        ady = 0.
      endif
      end
```

Complex Control Flow 3: Do while expressed with 2 gotos

```
C=============================================
C     for given x find y such that
C     0 = sqrt(y) - y + x
C     (with accuracy epsilon)
C     avoid endless loop
C=============================================
      SUBROUTINE MODEL( N, X, Y )
      implicit none
      INTEGER N
      REAL    X(N), Y
      real yold, epsilon
      integer i, imax
      parameter (epsilon=0.0001, imax=100)

      y = 2.
      i = 1
 100  continue
      yold = y
      y = sqrt(y) + x(1)
      if ( i .gt. imax) goto 200
      if ( abs(y-yold) .gt. epsilon) goto 100
      return
 200  continue
      print*,'bad luck: no convergence'
      END
```

Complex Control Flow 3

```
TAF log file:

...
=========================================
 semantic analysis
=========================================

=========================================
 normalization
=========================================

=========================================
 control flow analysis
=========================================
control flow analysis of subroutine model
 ==> irreducible control flow graph
...
```

TAF cannot normalise (yet)
–> User must normalise structure beforehand

Complex Control Flow 3: Normalised by hand: o.k for TAF

```
C=============================================
C     for given x find y such that
C     0 = sqrt(y) - y + x
C     (with accuracy epsilon)
C     avoid endless loop
C=============================================
      SUBROUTINE MODEL( N, X, Y )
      implicit none
      INTEGER N
      REAL    X(N), Y

      real yold, epsilon
      integer i, imax
      parameter (epsilon=0.0001, imax=100)

      y = 2.
      i = 1
 100  continue
      i = i + 1
      yold = y
      y = sqrt(y) + x(1)
      if ( abs(y-yold) .gt. epsilon .and. i .lt. imax) goto 100

      if (i .eq. imax) print*, 'bad luck: no convergence'

      END
```

Providing required values: Recomputation

Model Code:

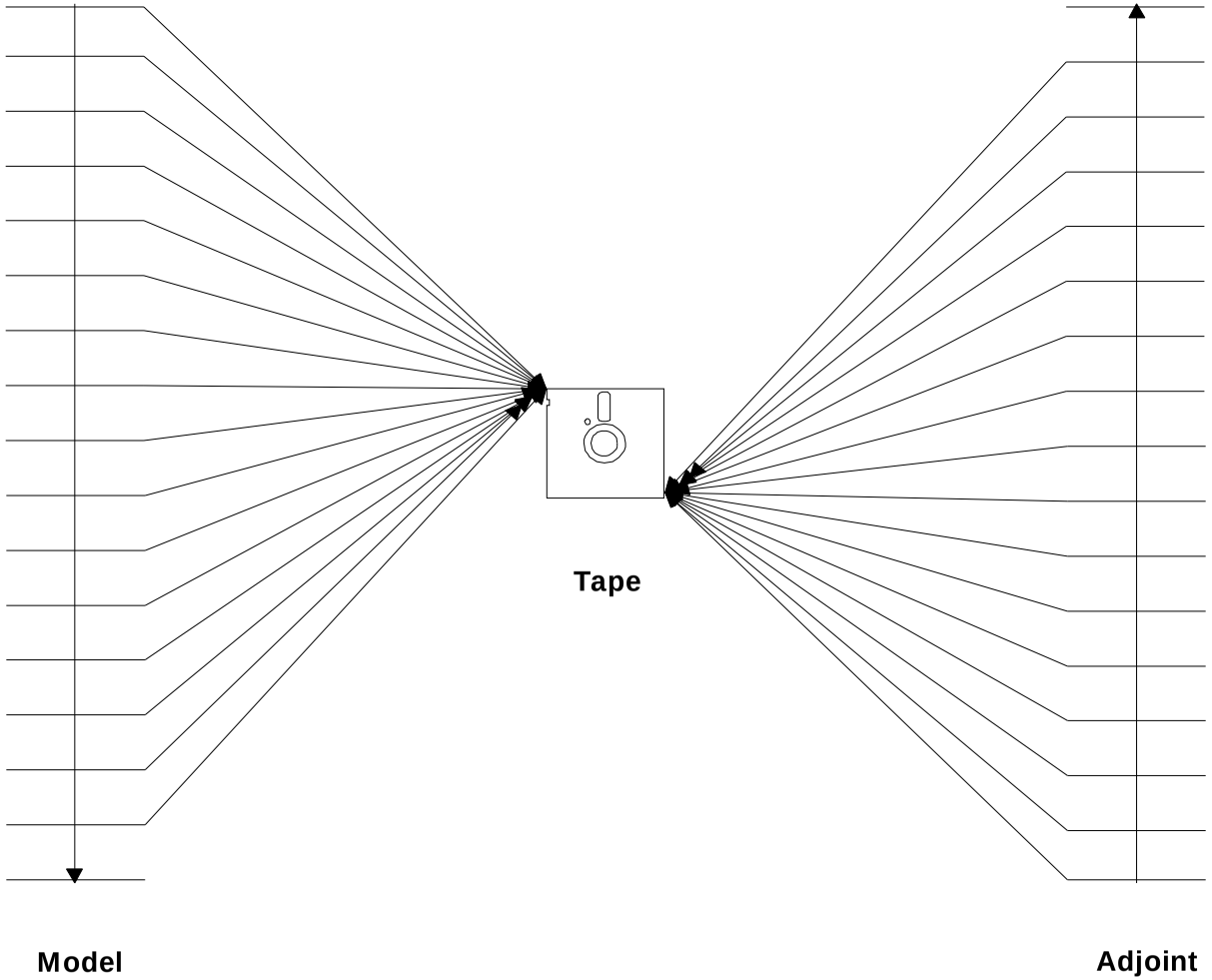
```
FC = 1.0
DO I = 1, N
  FC = FC * X(I)
ENDDO
```

Corresponding Adjoint Code:

```
do i = n, 1, -1
  fc = 1.
  do i2 = 1, i-1
    fc = fc*x(i2)
  end do
  adx(i) = adx(i)+adfc*fc
  adfc = adfc*x(i)
end do
```

By default TAF generates recomputations

Storing of required variables



Providing required values: Storing/Reading

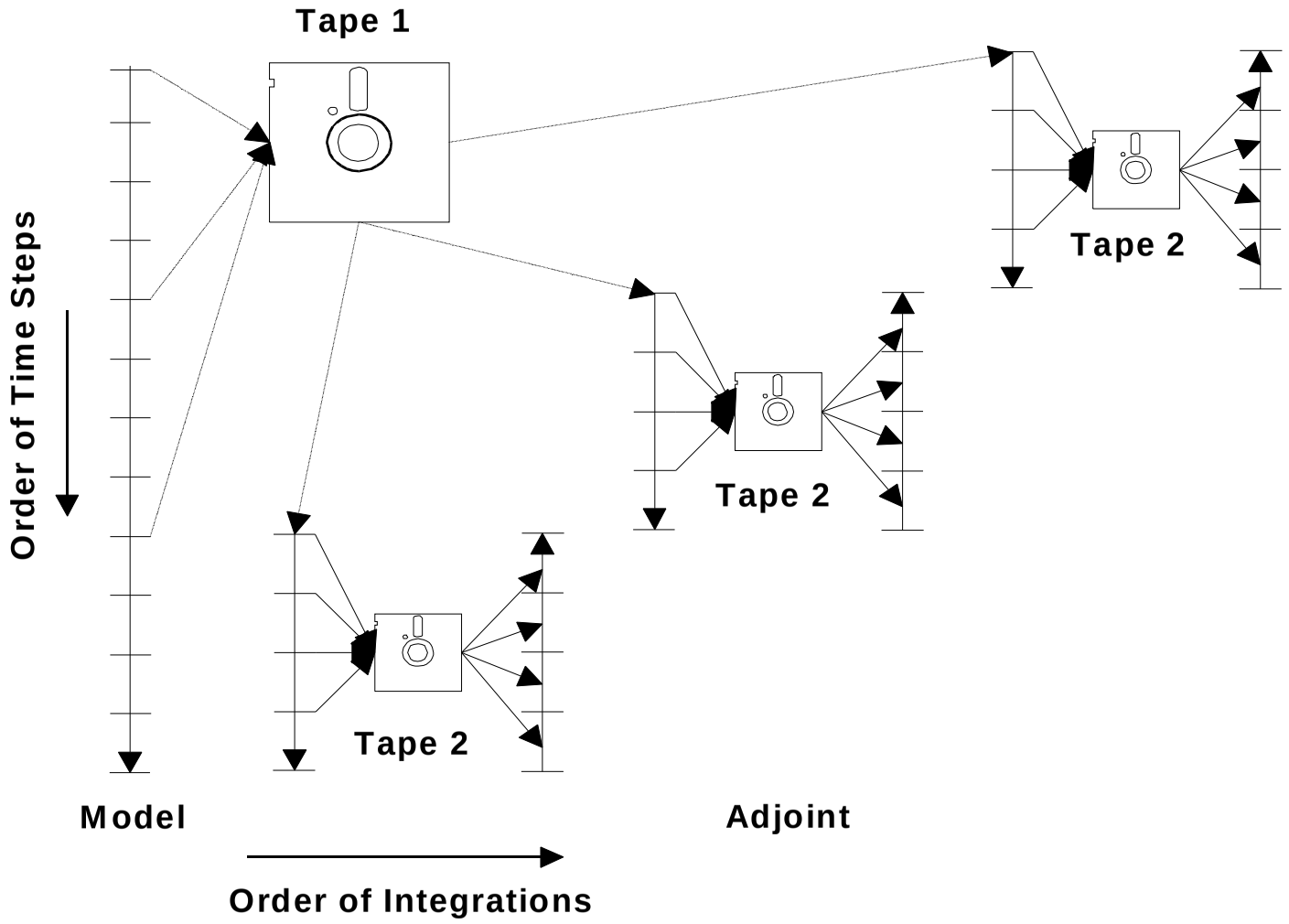
Model Code with directives:

```
C$TAF INIT mytape = „myfile“
      FC = 1.0
      DO I = 1, N
C$TAF STORE FC = mytape
        FC = FC * X(I)
      ENDDO
```

Corresponding Adjoint Code:

```
      open(unit=60,file='myfile_1_model_fc',ACCESS='DIRECT',RECL=1*4)
      fc = 1.
      do i = 1, n
        write(unit=60,REC=i) fc
        fc = fc*x(i)
      end do
C-----------------------------------------
C ADJOINT COMPUTATIONS
C-----------------------------------------
      do i = n, 1, -1
        read(unit=60,REC=i) fc
        adx(i) = adx(i)+adfc*fc
        adfc = adfc*x(i)
      end do
```

Checkpointing



Providing required values: Directives for checkpointing

```
C----------------------------------------
C define outer tape on file
C----------------------------------------
C$TAF INIT outtape = "checkpoints"

      FC = 1.0
      DO IOUT = 1, N/5
C----------------------------------------
C store variable FC on outer tape
C----------------------------------------
C$TAF STORE FC = outtape
C----------------------------------------
C define inner tape in dynamic memory
C----------------------------------------
C$TAF INIT intape = memory

         DO IIN = 1, 5
C----------------------------------------
C store variable FC on inner tape
C----------------------------------------
C$TAF STORE FC = intape
            I = IIN + (IOUT-1)*5
            FC = FC * X(I)
         ENDDO
      ENDDO
```

TAF Derivatives of Large Models

Model (Who)	Lines	Lang	TLM	ADM	Ckp	HES
NASA/NCAR (w. Todling & Lin)	87’000	F90	2.7	8.4	2 lev	
MOM3 (Galanti & Tziperman)	50’000	F77	Yes	4.0	2 lev	
MITGCM (ECCO Consortium)	100’000	F77	1.8	4.5	2 lev	11,0
BETHY (w. Knorr, Rayner, Scholze)	5’000	F90	1.5	3.6	2 lev	12.7
Nav.−Stokes−Solver (Hinze−Slawig)	1’000	F77		1.3	steady	
NSC2KE	3’400	F77		5.0	steady	

- Lines: total number of Fortran lines without comments
- Numbers for TLM and ADM give CPU time for (function + gradient) relative to forward model
- Number for HES gives CPU time for Hessian * vector
- 2 level checkpointing costs 1 additional model run

Performance compared to hand coded adjoints

Code	Hand	TAF/TAMC	relativ
EPT (MINPACK–2)	1.5	1.9	26%
GL1 (MINPACK–2)	1.5	1.7	13%
GL2 (MINPACK–2)	2.1	1.3	–40%
MSA (MINPACK–2)	1.75	1.6	–9%
PJB (MINPACK–2)	1.75	2.2	+25%
SSC (MINPACK–2)	1.18	1.13	–5%
Nav.–Stokes (H & S)	1.9	1.3	–30%

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

TAMC Adjoints of Large Models

Model (Who)	TLM	ADM	Checkp
PUMA (Blessing)	2.0	2.8	
MM5 (Nehrkorn et al.)	Yes	Yes	?
TM2 (Kaminski et al.)		3.4	2 level
TM3 (Roedenbeck et al.)		Yes	?
HCM (Eckert)	Yes	3.7	2 level
MOM3 (Galanti & Tziperman)		4.0	2 level
MITGCM (ECCO Consortium)	1.8	4.5	2 level
HOPE (Oldenborg et al.)		5–6	3 level
WAM (Hersbach)		3.7	2 level

- Numbers for TLM and ADM give CPU time for (function + gradient) relative to forward model
- 2 (3) level checkpointing costs 1 (2) additional model runs

TAF: Ongoing Development

- adapt TAF to new Fortran standards
  - Current Tasks: OpenMP (see Ralf's talk), MPI
  - Future Task:Fortran 2000
- extend TAF code analyses
- improve TAF algorithms
- support TAF users