Generating the tangent linear and adjoint models of the NASA/NCAR climate model's dynamical core by means of TAF

R. Giering, T. Kaminski, R. Todling, and S.−J. Lin

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

Overview of presentation

- Introduction to model
- General Approach
- Modifications to Model
- TAF Enhancements
- Handling of OpenMP
- Handling of MPI

Introduction to the model

- **NASA/NCAR climate model's dynamical core**
- **Formulation is based on work of Lin and Rood.**
- **All physical parameterizations (including the land surface model) are derived from the NCAR CCM 3.**
- **Written in Fortran 90, uses e.g.**
  - **modules**
  - **derived types**
  - **dynamic memory allocation**
- **Uses MPI−1 or MPI−2**
- **Uses OpenMP**

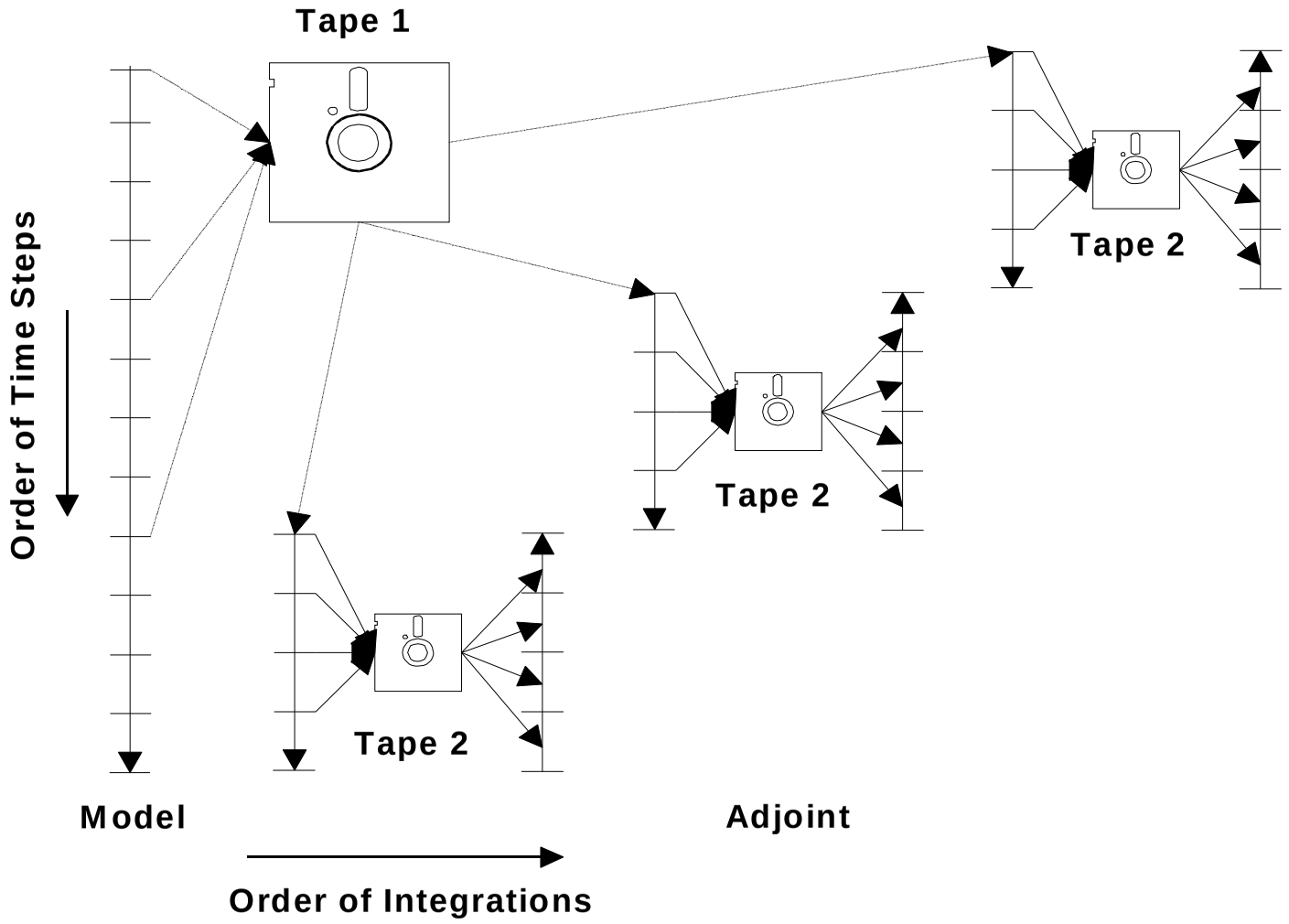
Approach

- **Generate TLM and ADM automatically by applying Automatic Differentiation tool TAF to model code**
- **Combination of**
  - **Modifications to the model code**
  - **Enhancements of TAF**
- **Assure / test that model output is not affected by modifications**
- **Assure / test correctness of TLM and ADM by checking against finite difference approximation**

General Modifications

- **Add code to write values of prognostic fields (winds, delta pressure, pot. temperature, moisture) after integration as reference fields – after further modifications, we check against those**
- **Function definition:**
  independent variables: to initialize prognostic fields (winds, moisture, surf. pressure, virt. temperature)
  dependent variable: cost function is difference between reference fields and simulated fields
- **Modular structure**
  - **initialisation**
  - **function evaluation**
  - **postprocessing**
- **Prepare corresponding set of driver programs for testing und running TLM and ADM**

Checkpointing



Providing required values: Checkpointing Scheme in stepon

```
!$taf init outer = static, nouter
!-------------------------------------------------------
! Beginning of basic time step loop
!-------------------------------------------------------
  do iouter = 1, nouter
!$taf store u3s,v3s,delp,q3,pt   = outer, key=iouter
!$taf store ns,pe,pk,pkz,ps      = outer, key=iouter
!$taf init inner = static, ninner
    do iinner = 1, ninner
      nstep = (iinner-1) + (iouter-1)*ninner
!$taf store delp,pe,pk,pkz,ps,pt,q3,u3s,v3s = inner, key=iinner
!$taf store omega,ns                        = inner, key=iinner
      call dynpkg_do ( ... )
!$taf store pe,pkz,pt,u3s,v3s,q3,t3,pk,omga  = inner, key=iinner
      if (ideal_phys) then
        call hswf_do
      endif
!$taf store u3s,v3s,pt,pe,pkz = inner, key=iinner
      call d_p_coupling ( ... )
      if ( .not. adiabatic ) then
!$taf store pe,phys_state,piln,pkz,q3 = inner, key=iinner
        call p_d_coupling ( ... )
      endif
! Ending time step logic
    end do   ! inner loop
  end do   ! outer loop
```

More Directives

- 57 TAF store directives, 25 TAF init directives
- TAF flow directives for subroutine rfftmlt (FFT):
  - TL and AD are hand written wrappers
  - reuse FFT code (with normalisation) in TLM / ADM (adjoint of FFT is inverse FFT)
- TAF loop directives to support TAF analysis
- TAF flow directives for black box routines
  - timing routines, allowing timing of TLM, ADM
  - I/O routines
  - initialization/allocation (deallocation) routines
    hand coded TL and AD routines
    to handle corresponding TL and AD objects

More Modifications: Complex control flow structures

- **Reverting control flow is complicated**
  => poses problem to ADM
- **These structures use statements like:**
  - **GOTO**
  - **ENTRY**
  - **CYCLE**
  - **EXIT**
  - **RETURN**
- **TAF replaces some structures automatically (code normalisaton) by equivalent structure**
- **Remainder must be replaced by user**
- **We rearranged code at two places to avoid structure with CYCLE + EXIT**

TAF Enhancements examples

- Active derived types
- Taping of allocatable arrays
- Taping where # of records not know at compile time
- Flow directive for internal (contained) procedures
- allow for keyword as module name (precision)

Test Results

```
================================================
 CHECK OF PERTUBATIONS USING eps = 0.100E-07
================================================

     I         x(i)  fc1-fc2/eps     GRAD(fc)   DIFFERENCE
     1 0.100000E-01 0.201703E+00 0.202144E+00 0.218435E-02
     2 0.100000E-01 0.272631E+00 0.272515E+00 0.428163E-03
     3 0.100000E-01 0.454125E+02 0.454125E+02 0.680532E-06
     4 0.100000E-01 0.391539E+03 0.391539E+03 0.672779E-06
     5 0.100000E-01 0.981409E+02 0.981394E+02 0.152572E-04

------------------------
-----  check adjoint  -----
------------------------
==========================================
 CHECK OF GRADIENTS USING eps = 0.100E-07
==========================================

     I         x(i)  fc1-fc2/eps     GRAD(fc) RELATIVE ERR
     1 0.100000E-01 0.201703E+00 0.202906E+00 0.592933E-02
     2 0.100000E-01 0.272631E+00 0.272452E+00 0.656707E-03
     3 0.100000E-01 0.454125E+02 0.454134E+02 0.194337E-04
     4 0.100000E-01 0.391539E+03 0.391534E+03 0.134529E-04
     5 0.100000E-01 0.981409E+02 0.981394E+02 0.152708E-04
```

Performance on Linux PC

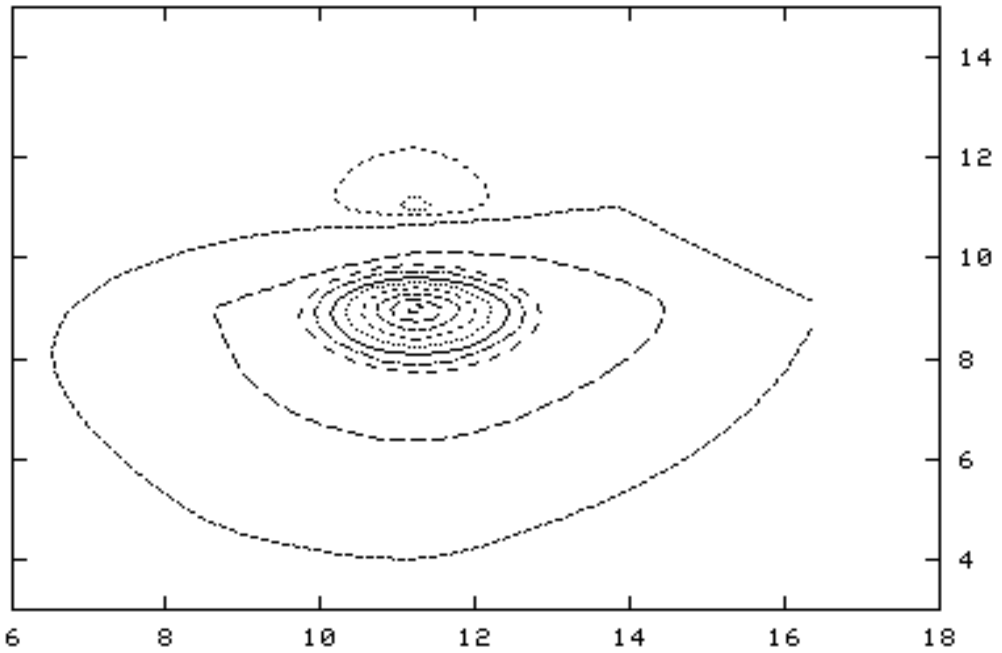
```
---------------------------
-----  time  tangent  -----
---------------------------
 ************************************************
   TIMING OUTPUT, niter =  1
 ************************************************
 control parameters N          :         5
 run time function             :     14.65300
 run time function + t.linear  :     39.09200
 rel. run time forward mode (FUNC+PERT)/FUNC          =      2.66785
 ************************************************

---------------------------
-----  time  adjoint  -----
---------------------------
 ************************************************
   TIMING OUTPUT, niter =  1
 ************************************************
 control parameters N          :         5
 run time function             :     14.62300
 run time function + adjoint   :    122.60900
 rel. run time reverse mode (FUNC+GRAD)/FUNC          =      8.38467
 ************************************************
```

Test Results

Sensitivity of potential temperature to zonal velocity

pt(12,10,9) to u3s(:,:,10)



Handling of Parallelization Strategy

- **Objective: TLM and ADM should have the same parallelization properties as the model itself**
- **Model uses OpenMP**
  => Enhanced TAF to handle relevant OpenMP automatically
- **Model uses MPI−1 and MPI−2**
  => Written TL and AD communication routines by hand

Handling of Parallelization OpenMP

Model uses only one directive:

!$omp parallel do

- **TAF analyzes the loop carried dependencies**
- **According to the dependencies TAF generates the proper !$omp directive for the adjoint loop and additional statements to preserve parallelism**
- **TAF uses the similar directive in the TL loop**

  shared –> AD,TLM shared,
  private –> AD,TLM private

Handling of Parallelization
OpenMP: Example in benergy ADM

```
!$omp  parallel do default(shared)    &
!$omp  private(i,j, k, u2, v2, t2, te_sp, te_np, veast)
      do 1000 k=1,km

... (bits of code removed)

! Compute dz; geo-potential increments
        do j=jfirst,jlast
           do i=1,im
              dz(i,j,k) = t2(i,j)*(pk(i,j,k+1)-pk(i,j,k))
           enddo
        enddo
1000  continue
```

- pk is shared
- two different loop pathes use the same memory location

=> in the adjoint same memory location of pk will be updated

=> race conflict if adjoint of pk is shared !

Handling of Parallelization OpenMP: Example in te_map ADM

```
adpkh(:,:,:) = adpk(:,:,:)
!$omp parallel do default(shared)
  shared(adcpt_t,addz,adpkh,adpkz,adpt,adte,adu,adv,cp,cpt_t,im,jfirs
  t,jlast,jm,jn2g0,js2g0,km,pk,&
&pkz,pt,u,v)
  private(adpk,adt2,adte_np,adte_sp,adu2,adv2,adveast,i,j,k,t2)
do k = 1, km
  adpk(:,:,:) = 0.
.....................(adjoint of the removed bits)
  do j = jfirst, jlast
    do i = 1, im
      adpk(i,j,k+1) = adpk(i,j,k+1)+addz(i,j,k)*t2(i,j)
      adpk(i,j,k  ) = adpk(i,j,k)-addz(i,j,k)*t2(i,j)
      adt2(i,j)     = adt2(i,j)+addz(i,j,k)*(pk(i,j,k+1)-pk(i,j,k))
      addz(i,j,k  ) = 0.
    end do
  end do
.....................(adjoint of the removed bits)
  !$omp critical  (adtaf)
  adpkh(:,:,:) = adpkh(:,:,:)+adpk(:,:,:)
  !$omp end critical (adtaf)
end do
adpk(:,:,:) = adpkh(:,:,:)
```

Handling of Parallelization
MPI: Strategy

- Model has wrapper routines (e.g. mp_send3d_ns) that call the respective MPI library routines (e.g. mpi_isend)
- Wrappers are encapsulated in module mod_comm
- Decision between MPI−1/2 happens in wrappers
- Construction of TL and AD at level of wrappers
- Inserting of TAF flow directives for wrappers
- TL and AD wrapper routines hand written
- TL and AD wrappers reuse model wrappers (easy to maintain)
- Handling of MPI−1 and MPI−2 at once
- Encapsulation helped a lot!

Handling of Parallelization
MPI: Example

Model Code calling sequence for wrapper in cd_core:

```
!------------------------------
! Send (u, v) on the way
!------------------------------
      call timing_on('Send_uv')
      call mp_barrier
      call mp_send3d_ns(im, jm, jfirst, jlast, 1, km, ng_d, ng_s, u,  1)
      call mp_send3d_ns(im, jm, jfirst, jlast, 1, km, ng_s, ng_d, v,  2)
      call timing_off('Send_uv')
```

TAF flow directives:

```
!$taf module mod_comm subroutine mp_send3d_ns  input = 1,2,3,4,5,6,7,8,9,10
!$taf module mod_comm subroutine mp_send3d_ns output = 9
!$taf module mod_comm subroutine mp_send3d_ns active = 9
!$taf module mod_comm subroutine mp_send3d_ns depend = 1,2,3,4,5,6,7,8,10
!$taf module mod_comm subroutine mp_send3d_ns adname = admp_send3d_ns
!$taf module mod_comm subroutine mp_send3d_ns ftlname = g_mp_send3d_ns
```

Parallelization: MPI example of TL

```
!=================================================
      subroutine g_mp_send3d_ns(im, jm, jfirst, jlast, kfirst, klast, &
                                ng_s, ng_n, q, g_q, iq)
!=================================================
      use mod_comm, only: mp_send3d_ns
      implicit none
      integer im, jm
      integer jfirst, jlast
      integer kfirst, klast
      integer ng_s      ! southern zones to ghost
      integer ng_n      ! noruthern zones to ghost
      real q(im,jfirst-ng_s:jlast+ng_n,kfirst:klast)
      real g_q(im,jfirst-ng_s:jlast+ng_n,kfirst:klast)
      integer iq
      call mp_send3d_ns( im, jm, jfirst, jlast, kfirst, klast, &
                               ng_s, ng_n, q, iq)
      call mp_send3d_ns( im, jm, jfirst, jlast, kfirst, klast, &
                               ng_s, ng_n, g_q, iq)
      end
```

Parallelization: MPI example of AD

```
!=====================================================
      subroutine admp_send3d_ns(im, jm, jfirst, jlast, kfirst, klast, &
                                ng_s, ng_n, adq, iq)
!=====================================================
      use mod_comm, only: mp_recv3d_ns
      implicit none
      integer im, jm
      integer jfirst, jlast
      integer kfirst, klast
      integer ng_s       ! southern zones to ghost
      integer ng_n       ! noruthern zones to ghost
      real q(im,jfirst-ng_s:jlast+ng_n,kfirst:klast)
      real adq(im,jfirst-ng_s:jlast+ng_n,kfirst:klast)
      integer iq
! local:
      real adq_h(im,jfirst-ng_s:jlast+ng_n,kfirst:klast)

      adq_h = 0.
      call mp_recv3d_ns( im, jm, jfirst, jlast, kfirst, klast, &
                                 ng_s, ng_n, adq_h, iq)
      adq = adq + adq_h
      end
```

Summary

- We applied TAF to generate the TLM and ADM for a GCM written in Fortran 90
- A number of modifications to the model code had to be carried out
- A few TAF enhancements were necessary
- The ADM includes an automatically generated checkpointing scheme
- Preserving OpenMP parallelism for the TLM and ADM could be handled automatically by TAF
- MPI parallelism was preserved via hand written TL and AD wrapper routines and TAF flow directives