P 6.18

MM5 ADJOINT DEVELOPMENT USING TAMC: EXPERIENCES WITH AN AUTOMATIC CODE GENERATOR

Thomas Nehrkorn*, George D. Modica, Mark Cerniglia
Atmospheric and Environmental Research, Inc. Lexington, MA

Frank H. Ruggiero
Space Vehicles Directorate, Air Force Research Laboratory, Hanscom Air Force Base, MA

John G. Michalakes
Argonne National Laboratory, Boulder, CO

Xiaolei Zou
Department of Meteorology, Florida State University, Tallahassee, FL

1. BACKGROUND

The four-dimensional variational (4DVAR) approach has been shown to provide benefits and advantages to NWP when compared with other methods of data assimilation. However, the computational demands of this approach far exceed the computer capacity available at a typical weather forecast center. As part of a project described in a companion paper at this conference (Ruggiero et al. 2001), we are developing a highly scalable version of the MM5 adjoint modeling system to allow its efficient execution on parallel distributed memory (DM) machines. To take advantage of the existing parallelization mechanisms (Michalakes, 2000) already in place for the latest release (Version 3.4) of the MM5 nonlinear model (NLM), the existing (Version 1, Zou et al. 1998) tangent linear (TLM) and adjoint model (ADJ) codes are also being updated to Version 3.4. In order to facilitate the code generation of the TLM and ADJ, we utilize the Tangent Linear and Adjoint Model Compiler. In Section 2 we describe the TAMC, its capabilities, and the procedure we followed to generate TLM and ADJ code. This is followed in Section 3 by a description of the tests used to check for the correctness of the TLM and ADJ. Finally, we present some details of our code development methodology in Section 4.

2. THE TANGENT LINEAR AND ADJOINT MODEL COMPILER (TAMC)

TAMC is a source-to-source translator that generates Fortran code for the TLM or ADJ from the Fortran code of the nonlinear model (NLM). The NLM source code must conform to the Fortran-77 standard, with some additional restrictions (some features of the Fortran-90 standard are also supported). TAMC was developed and tested by Giering and Kaminski (1998), and it has been used in oceanographic and other modeling applications (e.g., Kaminski et al. 1996). The NLM derivatives are computed in the reverse mode (to create the ADJ) or in the forward mode (TLM). TAMC normalizes the source code and applies a control flow analysis. TAMC also applies an intraprocedural data dependence and an interprocedural data flow analysis. Given the independent and dependent variables of the specified top-level routine, TAMC determines all active routines and variables and produces derivative code only for those.

It would be possible to incorporate the TAMC as part of the NLM compilation process, requiring the maintenance of just the NLM code. This requires, however, that the NLM code be modified

* *Corresponding author address:* Thomas Nehrkorn, AER Inc., 131 Hartwell Ave., Lexington, MA, 02421-3126; email: trn@aer.com

as needed to result in generation of correct TLM and adjoint code by TAMC. For this project, we have found that amount of code modification to be prohibitive. For the development of the MM5 adjoint, we have chosen instead to use TAMC as a development tool only, and separately maintain the TLM and adjoint versions of the model code. This approach makes it possible to minimize changes to the MM5 code as supported by NCAR, but it requires a mixture of manual and automatic code generation. Compared to manual coding only, this method substantially reduces the time for coding and error debugging.

3. CODE GENERATION AND TESTING

As part of the TLM and adjoint model development, the MM5 NLM code is temporarily modified to remove some nonstandard features (primarily, the use of Fortran-77 pointer variables) before being passed to TAMC. The output generated by TAMC is then hand-checked by comparison against the existing, hand-coded TLM and adjoint Version 1 codes, taking into account differences between the Version 1 and 3.4 NLM codes. Manual corrections are made as needed.

Both the TLM and adjoint are tested for correctness. We use the standard comparison of the TLM and finite difference NLM gradients to check for correctness of the TLM:

$$\frac{[J(\mathbf{X}+\delta\mathbf{X})-J(\mathbf{X})]}{\delta J(\mathbf{X},\delta\mathbf{X})}=1 \qquad \lim \delta\mathbf{X}\to 0 \tag{1}$$

where $J(X+\delta X)$ and $J(X)$ are from the NLM, and $\delta J(X, \delta X)$ is from the TLM. The definition of the adjoint is used to check for consistency between the TLM and adjoint:

$$\langle \mathbf{y}, L\mathbf{x}\rangle = \langle L^{*}\mathbf{y}, \mathbf{x}\rangle \tag{2}$$

where x is the TLM input, y = $L(x)$ is the TLM output, and L, L^* are the TLM and ADJ, respectively. This testing is performed for individual subroutines as well as the complete model integration.

4. DISCUSSION

Overall, there was remarkable agreement between the TAMC generated derivative (TLM and adjoint) code and the hand-coded versions, not only in the assignment statements, but also in the choice between storing and recalculating intermediate NLM quantities. Almost all of the problems encountered by TAMC have to do with the recomputation of the NLM, not with the derivative code itself. The most common types were caused by known limitations of TAMC:

- complicated control flow with goto statements ("spaghetti code")
- dependencies on array elements.

These are illustrated in the following examples.

4.1 *An Example Of Incorrect NLM Dependency Analysis:* **BDYRST**

In this example, the dependency of TAMC is incorrect because TAMC considers dependencies of entire arrays, not individual elements. The NLM code for this subroutine contains if-statements that depend on the assignments to TA in the first two do-loops (over j and i):

```
do nb1=1,lb
 nb=nb1
 is=nb1
 js=nb1
 ie=iend+1-nb1
 je=jend+1-nb1
 do k=1,kl
  do j=nb+1,je-1
   ta(is,j,k)=tsb(j,k,nb)+dtbc*tten(is,j,k)
   ta(ie,j,k)=tnb(j,k,nb)+dtbc*tten(ie,j,k)
  enddo
  do i=nb,ie
   ta(i,js,k)=twb(i,k,nb)+dtbc*tten(i,js,k)
   ta(i,je,k)=teb(i,k,nb)+dtbc*tten(i,je,k)
  enddo
  do j=nb+1,je-1
   if(ta(is,j,k).lt.epsi)ta(is,j,k)=epsi
   if(ta(ie,j,k).lt.epsi)ta(ie,j,k)=epsi
  enddo
  do i=nb,ie
   if(ta(i,js,k).lt.epsi)ta(i,js,k)=epsi
   if(ta(i,je,k).lt.epsi)ta(i,je,k)=epsi
  enddo
 enddo
enddo
```

If the "-pure" option was selected (in which NLM code is only generated if needed for the TLM), the TLM code generated by TAMC omitted the first recalculation of the TA variable. Without this option, the following correct code was generated (the lines omitted in the -pure version are highlighted):

```
do nb1 = 1, lb
 nb = nb1
 is = nb1
 js = nb1
 ie = iend+1-nb1
 je = jend+1-nb1
 do k = 1, kl
  do j = nb+1, je-1
   g_ta(is,j,k) = g_tsb(j,k,nb)+
                  g_tten(is,j,k) * dtbc
   ta(is,j,k) = tsb(j,k,nb)+dtbc*tten(is,j,k)
   g_ta(ie,j,k) = g_tnb(j,k,nb)+
                  g_tten(ie,j,k)*dtbc
   ta(ie,j,k) = tnb(j,k,nb)+dtbc*tten(ie,j,k)
  end do
  do i = nb, ie
   g_ta(i,js,k) = g_tten(i,js,k)*dtbc
                  g_twb(i,k,nb)
   ta(i,js,k) = twb(i,k,nb)+dtbc*tten(i,js,k)
   g_ta(i,je,k) = g_teb(i,k,nb)+g_tten(i,je,k)
                  *dtbc
   ta(i,je,k) = teb(i,k,nb)+dtbc*tten(i,je,k)
  end do
  do j = nb+1, je-1
  if (ta(is,j,k) .lt. epsi) then
   g_ta(is,j,k) = 0.
   ta(is,j,k) = epsi
  endif
  if (ta(ie,j,k) .lt. epsi) then
   g_ta(ie,j,k) = 0.
   ta(ie,j,k) = epsi
  endif
  end do
  do i = nb, ie
  if (ta(i,js,k) .lt. epsi) then
   g_ta(i,js,k) = 0.
   ta(i,js,k) = epsi
  endif
  if (ta(i,je,k) .lt. epsi) then
   g_ta(i,je,k) = 0.
   ta(i,je,k) = epsi
  endif
  end do
 end do
end do
```

4.2 *An Example Of Incorrect NLM Recomputation:* **CUPARA2**

In this example, the errors in the NLM code were generated by goto statements characteristic of older Fortran codes, as found in the Kuo cumulus parameterization scheme of the MM5. The NLM contained the following code segments, in which the vertical loop limits for the cloud computations are derived by a computation of cloud base and height. TAMC encountered two problems in this case: (1) it generated unnecessary TLM code of the cloud base and top (these are more properly considered the NLM trajectory about which to linearize), and (2) crucial goto-statements (highlighted below) were missing, which led to incorrect loop limits.

```
c$$$ The preceding code computes the lifting
c$$$ condensation level sigma (SIGLCL) and the
c$$$ cloud and environmental values of
c$$$ saturation equivalent potential
c$$$ temperature (EQTM and SEQT)

    DO 220 K=1,KL
     IF(A(K).GE.SIGLCL)GOTO 230
220 CONTINUE
230 CONTINUE
kbase = k
if (kbase .gt. kl) then
  kbase = kl
endif
do kk = 1, kbase
  k = kbase+1-kk
  DEQT=SEQT(K)-EQTM
  IF(DEQT.GT.DLT)GOTO 270
end do
270 continue
ktop = k
```

These errors were corrected manually.

4.3 *An Example Of Inefficient NLM Recomputations:* **TRANSM**

In many of the MM5 physics subroutines all computations are performed independently for each vertical column, but are contained inside a loop over a horizontal grid index (i) for computational efficiency. In some cases the adjoint code generated by TAMC reversed the order of this i-loop, and either inserted nested loops over i, or used local storage to recompute or restore the NLM for all previous horizontal gridpoints. For example, in the following NLM code atmospheric radiative transmissivities are computed from, among others, column precipitable water (PRW) computed previously:

```
ilx = iyy-1
do i = 1, ilx
 prw(i,j) = amin1(prw(i,j),4.99)
 prw(i,j) = amax1(prw(i,j),0.01)
end do
do i = 1, ilx
c$$$ The following computes the output from
prw
(code not shown)
end do
```

In the above, i and j are both horizontal grid indices. The TAMC generated code contains storage and recomputations for all horizontal gridpoints (unnecessary statements highlighted):

```
do ip2 = 1, mjx
 do ip1 = 1, mix
  prwh(ip1,ip2) = prw(ip1,ip2)
 end do
end do
ilx = iyy-1
do i = 1, ilx
 prw(i,j) = amin1(prw(i,j),4.99)
 prw(i,j) = amax1(prw(i,j),0.01)
end do
do i = 1, ilx
c$$$ The following computes the
c$$$ adjoint of prw
(code not shown)
end do
do ip2 = 1, mjx
 do ip1 = 1, mix
  prw(ip1,ip2) = prwh(ip1,ip2)
 end do
end do
do i = 1, ilx
 prw(i,j) = amin1(prw(i,j),4.99)
 a_prw(i,j) = a_prw(i,j)*(0.5+
              sign(0.5,prw(i,j)-0.01))
 do ip2 = 1, mjx
  do ip1 = 1, mix
   prw(ip1,ip2) = prwh(ip1,ip2)
  end do
 end do
 a_prw(i,j) = a_prw(i,j)*(0.5+
              sign(0.5,4.99-prw(i,j)))
end do
```

This code was manually simplified as follows:

```
do ip1 = 1, mix
 prwh(ip1) = prw(ip1,j)
end do
ilx = iyy-1
do i = 1, ilx
 prw(i,j) = amin1(prw(i,j),4.99)
 prw(i,j) = amax1(prw(i,j),0.01)
end do
do i = 1, ilx
c$$$ The following computes computes the
c$$$ adjoint of prw
(code not shown)
end do
do i = 1, ilx
c$$$  Only restore prw for current i:
 prw(i,j) = prwh(i)
 prw(i,j) = amin1(prw(i,j),4.99)
 a_prw(i,j) = a_prw(i,j)*(0.5+
              sign(0.5,prw(i,j)-0.01))
c$$$  Only restore prw for current i:
 prw(i,j) = prwh(i)
 a_prw(i,j) = a_prw(i,j)*(0.5+
              sign(0.5,4.99-prw(i,j)))
end do
```

5. CONCLUSIONS

The TAMC derivative code generator was successfully applied to the MM5 (v3.4). For this model code, for which both tangent linear and adjoint source codes were available for a previous version, we found that TAMC was best used as a development tool, with separately maintained TLM and adjoint versions of the source code. This approach makes it possible to minimize changes to the MM5 code as supported by NCAR, but it requires a mixture of manual and automatic code generation. Manual corrections and simplifications of the generated code were necessary in some cases, mostly involving recomputations of the NLM.

Acknowledgements: This work was supported by the Department of Defense (DoD) High Performance Computing (HPC) Modernization Office's Common High Performance Computing Software Support Initiative. Additional support was provided from the DoD Major Shared Resource Center at Wright-Patterson AFB for time on the IBM SP3.

REFERENCES

Giering, R. and T. Kaminski, 1998: ACM Trans. on Math. Software, **24(4)**, 437-474.

Kaminski, T., R. Giering, and M. Heimann: Phys. and Chem. of the Earth, **21(5-6)**, 457-462.

J.G. Michalakes, 2000: Scientific Programming, **8**, 5-12.

Ruggiero, F.H., G.D. Modica, T. Nehrkorn, M. Cerniglia, J. Michalakes, X. Zou, 2001: A MM5-based four-dimensional variational analysis system developed for distributed memory computers. *Preprints*, *14th Conf. on Numerical Weather Prediction*, 30 July-2 August, 2001, Ft. Lauderdale, FL, American Meteorological Society (this volume).

Zou, X., W. Huang, and Q.~Xiao, 1997: A User's Guide to the MM5 adjoint modeling system. NCAR TN 437+IA.