CAPACITES ACTUELLES DE LA DIFFERENTIATION AUTOMATIQUE: L'ADJOINT D'OPA 9.0 PAR TAPENADE

Bruno Ferron* et Laurent Hascoët**

*IFREMER, Laboratoire de Physique des Océans, 29280 PLOUZANE, France

**INRIA, Projet Tropics, 06901 SOPHIA-ANTIPOLIS, France

Résumé: L'assimilation variationnelle de données repose sur l'utilisation de codes adjoints. Ces codes peuvent être soit écrits à la main, soit générés par Différentiation Automatique (DA) du programme de résolution des équations directes initiales. Les outils de DA existent depuis une vingtaine d'années. Ils ont atteint un niveau de maturité suffisant pour produire automatiquement en quelques minutes des codes adjoints d'une qualité comparable à ceux écrits à la main. Après un bref rappel des principes de la différentiation adjointe ou inverse, nous décrivons l'état de l'art de ce domaine de la DA. Nous mettons l'accent sur l'avancement de l'outil de DA « TAPENADE » développé par l'équipe TROPICS de l'INRIA. Nous illustrons les capacités de la DA et de TAPENADE par la dérivation de l'adjoint du code d'océanographie OPA 9.0.

1 INTRODUCTION

Notre but est de montrer l'intérêt de la Différentiation Automatique (DA) pour obtenir rapidement des codes adjoints efficaces, utilisables pour l'assimilation variationnelle des données. Les codes adjoints nécessaires à l'assimilation peuvent être écrits à la main, par discrétisation et résolution des équations adjointes. Ce processus est long, coûteux et source potentielle d'erreurs. De plus, un code adjoint étant construit par référence à un code de simulation donné, dit code initial ou « direct », il doit être remis à jour après chaque modification ou raffinement du code direct. Cela survient fréquemment dans le cycle de vie d'un gros code de simulation numérique, en particulier en météorologie ou océanographie, que ce soit pour intégrer de nouvelles équations physiques ou pour perfectionner les algorithmes.

La DA, présentée en section 2, est une technique logicielle qui modifie un programme source calculant une fonction, pour qu'il calcule les dérivées analytiques de cette fonction. Il existe plusieurs modes de DA, selon le type de dérivées demandées. En particulier le mode « inverse » permet de calculer des gradients par une stratégie qui est l'analogue discret de la méthode des équations adjointes. Pour certains modèles simples, on peut même montrer l'identité des codes résultant de la stratégie consistant à écrire les équations adjointes puis à les résoudre par programme d'une part, et d'autre part de la stratégie consistant à différentier en mode inverse le programme qui résout les équations directes. Nous préconisons donc d'employer un outil de DA en mode inverse pour produire à moindres frais les codes adjoints pour l'assimilation variationnelle.

Les outils de DA existent depuis une vingtaine d'années. Cependant, le mode inverse présente des caractéristiques qui rendent particulièrement délicate la construction automatique d'un code différentié efficace. Le problème central est la reconstruction pour l'adjoint des états successifs du calcul direct, et ce dans l'ordre inverse. Les outils proposant un mode inverse efficace sont donc apparus plus tardivement. Il reste d'ailleurs un certain nombre de problèmes ouverts, objets de recherches actives. Nous présentons ces recherches en section 3, et leurs conséquences pour les outils de DA, en particulier TAPENADE.

TAPENADE [9] est l'outil de DA développé par l'équipe TROPICS de l'INRIA. Son mode inverse est utilisé intensivement par exemple en optimisation de forme [5] pour des écoulements stationnaires en mécanique des fluides. L'adjoint en assimilation est une application d'autant plus motivante qu'il s'agit de différentier une application instationnaire, un ordre de grandeur plus complexe. TAPENADE a été récemment étendu de FORTRAN 77 a FORTRAN 95, et inclut maintenant des analyses et des fonctionnalités sophistiquées pour le mode inverse. La construction par TAPENADE de l'adjoint du code OPA 9.0 est un objectif ambitieux mais atteignable. La section 4 présente les premiers résultats de ce travail.

2 PRINCIPES DE LA DIFFERENTIATION AUTOMATIQUE (DA) EN MODE INVERSE

Le principe de la DA [2,4,7] est d'appliquer la règle de différentiation des fonctions composées, sur un programme considéré comme la composition des fonctions élémentaires calculées par ses instructions successives. L'enjeu est de calculer des dérivées analytiques plutôt que les approximations calculées par différences divisées. A chaque instruction du programme initial calculant un résultat y en fonction de variables x_i, on ajoute de nouvelles opérations qui calculent la différentielle de y en fonction des x_i et de leurs différentielles. On obtient naturellement le mode « tangent » de la DA, qui calcule en fait des dérivées directionnelles. Cette augmentation du programme peut être réalisée facilement, par exemple par la technique logicielle de surcharge des opérateurs. C'est l'option choisie par des outils comme ADOL-C, mais elle se prête mal au mode inverse. Les outils comme ADIFOR [3], TAF [6] ou TAPENADE [9] préfèrent analyser globalement le programme et reconstruire totalement un programme différentié. En fait ce sont des compilateurs un peu spéciaux, qui produisent un nouveau code source.

Etant donné un programme P calculant, à partir d'un argument X dans R^n, un résultat Y=F(X) dans R^m, le mode inverse produit un programme « adjoint » Pb qui calcule, à partir de X et d'un vecteur ligne Yb de pondération, le gradient Yb × F'(X) de la fonction scalaire Yb × Y. Si l'on considère pour simplifier que P est la séquence d'instructions $\{I_1; I_2; \ldots I_{p-1}; I_p;\}$, la fonction F est en fait la composition $f_p \circ f_{p-1} \circ \ldots \circ f_2 \circ f_1$ des

fonctions élémentaires f_k implémentées par les I_k. Appelons $X_0=X$ et $X_k=f_k(X_{k-1})$ les valeurs successives de l'ensemble des variables du programme. Le gradient recherché s'écrit:

$$Yb \times F'(X) = Yb \times f_p'(X_{p-1}) \times f_{p-1}'(X_{p-2}) \times ... \times f_2'(X_1) \times f_1'(X_0)$$

Le programme adjoint doit calculer ce gradient. Comme Yb est un vecteur, il est bien plus efficace d'effectuer les produits de gauche à droite, ce qui motive le nom «inverse». Grâce à cette observation, le mode inverse est capable en théorie de calculer le gradient en un temps qui est un petit multiple du temps de P. Par comparaison, le calcul du gradient par l'intermédiaire des dérivées directionnelles du mode « tangent » coûterait un temps proportionnel à n, la dimension de X.

Il y a pourtant une difficulté majeure: le mode inverse évalue en premier $f_p'(X_{p-1})$, qui utilise l'état X_{p-1}, puis $f_{p-1}'(X_{p-2})$, qui utilise X_{p-2}, jusqu'à $f_1'(X_0)$, qui utilise X_0. On doit donc reconstruire les valeurs des variables dans l'ordre inverse de leur calcul par P. Deux stratégies extrêmes sont envisageables:

- « Recalculer Tout »: repartir autant de fois que nécessaire de l'état X_0 stocké auparavant. C'est la stratégie de TAF, symbolisée sur la gauche de la figure 1. Son coût principal est le nombre de recalculs nécessaires.
- « Stocker Tout »: mémoriser les différences entre les états successifs X_{k-1} et X_k, lors d'une exécution préliminaire de P. C'est la stratégie de TAPENADE et d'ADIFOR, symbolisée sur la droite figure 1. Son coût principal est la quantité de mémoire nécessaire.

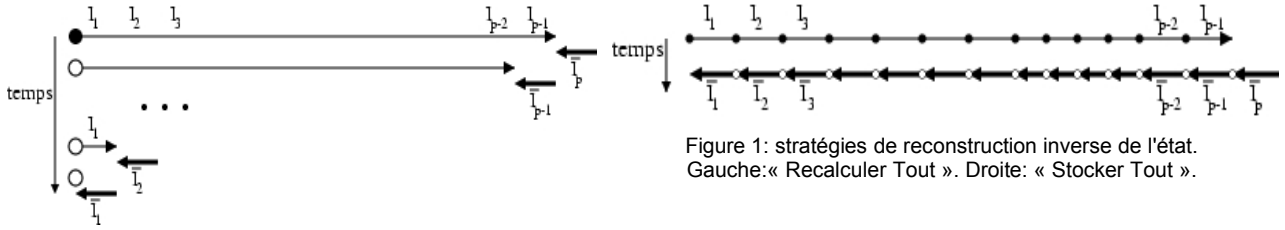

Figure 1: stratégies de reconstruction inverse de l'état. Gauche:« Recalculer Tout ». Droite: « Stocker Tout ».

Dans la pratique, aucune de ces deux stratégies n'est réaliste pour un gros programme. On doit employer un compromis stockage/recalcul nommé « Checkpointing », que la figure 2 schématise dans le contexte « Stocker Tout ». Globalement, l'exécution de Pb symbolisée à droite de la figure 1 consiste en une « passe avant », exécution de P augmenté des stockages nécessaires, suivie d'une « passe arrière » qui calcule les dérivées en utilisant les valeurs stockées. Un ckeckpoint est un fragment C du programme P pour lequel la passe avant n'effectue aucun stockage. Lorsque la passe arrière atteint le checkpoint, on relance une exécution de C avec stockage, et ainsi la passe arrière peut reprendre. Au prix d'une double exécution de C et de la mémorisation de l'état au début de C (le « snapshot »), on a réduit la quantité maximale de stockage utilisé à l'issue de la passe avant.

Figure 2: application du « Checkpointing » au fragment C en mode « Stocker Tout ».

3 L'AVANCEMENT DE TAPENADE ET DES AUTRES OUTILS DE DA

On l'a vu, le problème principal du mode inverse est la difficulté à reconstruire les états mémoire dans l'ordre inverse. D'autres problèmes sont étudiés, comme l'ordre optimal de calcul des expressions dérivées qui minimise les calculs redondants. Enfin un aspect important est l'adaptation du modèle de la DA à la structure particulière d'algorithmes numériques fréquents. Ce dernier aspect motive des collaborations étroites entre numériciens et informaticiens. Les équipes de recherche explorent plusieurs pistes et nous en donnons ici un résumé partiel, centré sur l'intérêt des analyses statiques globales du programme initial.

L'analyse d'activité détecte les variables « actives » qui dépendent de manière différentiable des entrées X et qui influencent les résultats Y. Seules les dérivées des variables actives doivent être calculées. Toutes les autres dérivées sont nulles ou inutiles et doivent être simplifiées dans Pb. Les outils ADIFOR, TAF et TAPENADE implémentent cette analyse, qui ne peut exister en revanche dans le cadre d'outils par surcharge. Les analyses statiques sont des algorithmes coûteux qui parcourent la globalité du programme P et reposent sur des représentations internes performantes (graphes d'appel, graphes de flot de contrôle) développées pour les compilateurs.

En mode inverse, dans le cadre « Stocker Tout », l'analyse « To Be Recorded » [8] limite le stockage aux valeurs effectivement utilisées dans les dérivées. On exploite le fait que la dérivée d'un opération linéaire n'utilise pas les valeurs des opérandes. On exploite également le fait que l'on ne demande pas au programme adjoint Pb la valeur du résultat Y de P. Par conséquent, on peut supprimer certaines instructions de P de la passe avant de Pb. C'est le but de l'analyse du code « mort pour l'adjoint » de TAPENADE. Dans le cadre « Recalculer Tout », l'algorithme « Efficient Recomputation » de TAF effectue des analyses similaires pour simplifier au maximum les nombreuses passes avant visibles a gauche de la figure 1. L'objectif d'un cadre commun pour hybrider recalcul et stockage de manière optimale est encore lointain.

Les tactiques précédentes ne peuvent que diminuer le coût calcul ou mémoire d'un facteur constant. Cela n'est pas suffisant pour les gros programmes. La solution passe par l'imbrication récursive des checkpoints. De cette manière, on peut limiter l'augmentation des coûts d'exécution du code inverse au

logarithme de la taille du programme initial, et on a montré qu'il s'agit d'un optimum dans un cas modèle. Trouver le schéma optimal de checkpoints imbriqués pour un programme arbitraire est encore un problème de recherche. TAF propose à l'utilisateur final de placer les checkpoints et de définir les snapshots lui-même. TAPENADE place systématiquement un checkpoint à chaque appel de procédure, mais permet à un utilisateur connaissant bien le code de supprimer certains d'entre eux. Les snapshots sont déterminés automatiquement, parfois au prix de sur-approximations conservatives.

Ces stratégies ont permis aux outils de DA de produire les adjoints de codes de plus en plus conséquents, et en particulier des simulations instationnaires au grand nombre de pas de temps. Les chercheurs de Dresde qui développent ADOL-C sont en pointe sur les schémas de checkpointing optimaux de boucles itératives, que leur nombre d'itérations soit statique ou dynamique. Leurs résultats de recherche sont transposés progressivement dans les autres outils. Tous ces travaux rendent tout à fait compétitif l'usage routinier de la DA pour produire les adjoints en assimilation de données en sciences de la Terre.

D'un point de vue plus pratique, les outils de DA doivent suivre l'évolution des langages de programmation. De plus en plus de codes migrent vers FORTRAN 95 ou C++. Rares sont les nouveaux codes écrits en FORTRAN 77. Parmi les conséquences pratiques pour les outils de DA, citons l'analyse des destinations possibles des pointeurs, et le problème ouvert du traitement des allocations dynamiques en mode inverse. Citons aussi la Différentiation inverse des instructions de tableaux en FORTRAN 95 et la gestion des types structurés, modules et classes. TAF et TAPENADE traitent tous deux l'essentiel de FORTRAN 95, et des versions destinées à C sont en développement.

Code initial:

```
MODULE divcur
 USE lbclnk  ! ocean lateral...
CONTAINS
 SUBROUTINE div_cur( kt )
 ...
 DO jk = 1, jpkm1
  hdivb(:,jk) = hdivn(:,jk)
  DO ji = 2, jpim1
   hdivn(ji,jk)=(e2u(ji)*un(ji,jk)-...)
                / (e1t(ji)*e2t(ji))
  END DO
 END DO
 CALL lbc_lnk( hdivn, 'T', 1._wp )
 END SUBROUTINE div_cur
END MODULE divcur
```

Code Adjoint construit par TAPENADE:

```
MODULE DIVCUR_B
 USE lbclnk_b
CONTAINS
 SUBROUTINE DIV_CUR_B(kt)
 ...
 CALL LBC_LNK_B(hdivn,hdivnb,'T',1._wp)
 DO jk=jpkm1,1,-1
  DO ji=jpim1,2,-1
   tempb=hdivnb(ji,jk)/(e1t(ji)*e2t(ji))
   unb(ji,jk) = unb(ji,jk)+e2u(ji)*tempb
   unb(ji-1,jk) = ... *tempb
   hdivnb(ji,jk) = 0.0
  END DO
  hdivnb(:,jk) = hdivnb(:,jk)+hdivbb(:,jk)
  hdivbb(:,jk) = 0.0
 END DO
 END SUBROUTINE DIV_CUR_B
END MODULE DIVCUR_B
```

Figure 3: adjoint d'un partie (simplifiée) de OPA 9.0 par TAPENADE. Les éléments dérivés sont terminés par un «b». La structure modulaire et les notations de tableaux sont préservées. L'analyse d'activité a supprimé certaines dérivées (`e2t`, `e2u`...), et l'analyse du code mort pour l'adjoint a permis de supprimer l'essentiel de la passe avant.

4 L'ADJOINT D'OPA 9.0 PAR TAPENADE

Nous présentons ici les résultats de la différentiation automatique d'une configuration d'OPA 9.0 par TAPENADE. Le but est de produire automatiquement un code adjoint et de valider les gradients obtenus. Nous décrivons le processus de différentiation puis nous montrons les performances en temps et mémoire de la différentiation par TAPENADE et surtout du code adjoint lui-même. OPA [1] est un modèle de circulation océanique générale développé principalement par le laboratoire LOCEAN à Paris VI. Dans sa version 8.0, en FORTRAN 77, il comporte un modèle adjoint qui a été développé à la main. La nouvelle version 9.0 d'OPA a été réécrite en FORTRAN 95 pour tirer parti de la modularité et des opérations « vectorielles » sur les tableaux. L'écriture manuelle de l'adjoint d'OPA 9.0 représenterait un énorme effort. Nous voulons montrer qu'on peut contourner cette difficulté grâce à la DA.

Nous avons choisi la configuration particulière d'OPA appelée « GYRE » développée par Marina Levy. Il s'agit d'un bassin rectangulaire fermé, forcé uniquement par le vent qui produit une double gyre océanique. Le gradient obtenu grâce à TAPENADE (figure 4) montre bien l'influence des ondes océaniques rapides situées à distance 20 jours plus tôt, sur le flux de chaleur mesuré au nord, ainsi que l'influence de l'advection au voisinage de la zone de mesure.

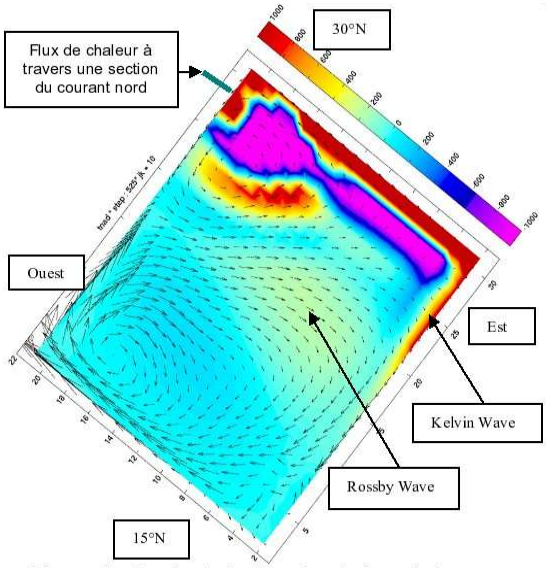

Figure 4: Gradient du flux de chaleur à travers une section du courant nord, par rapport au champ de températures à 300m, 20 jours auparavant.

Pour cette première expérience de différentiation automatique, la physique du code direct a été choisie résolument simple: équation d'état linéaire en température et salinité, diffusion verticale constante, surface libre, schéma d'advection centré, diffusion et viscosité horizontales Laplaciennes, free-slip latéral et friction linéaire au fond. L'objectif était de faciliter les inévitables aller-retours de mise au point de la différentiation. Il s'agit néanmoins d'une simulation significative. Le domaine 3D est discrétise en 32×22×31 mailles (1° de résolution horizontale) et la simulation dure 4320 pas de temps. Les nombres réels sont codés sur 8 octets. A l'exécution, cette simulation occupe une place de 86 Mo et dure 92 secondes sur un PC Pentium à 2 GHz.

La différentiation proprement dite par TAPENADE prend 20 secondes sur ce même PC. Ce temps est à rapprocher de la taille du code initial, à peu près 40 000 lignes de FORTRAN95. La figure 5 montre la validation de l'adjoint produit par TAPENADE par la méthode dite du produit scalaire: le code d'origine évaluant Y=F(X), on choisit une direction arbitraire Xd dans l'espace d'entrée. Dans cette direction, on calcule la dérivée directionnelle Yd = F'(X) × Xd de deux manières différentes: d'une part par différences divisées, avec un epsilon de 10^{-7}, d'autre part par un code différentié tangent produit par Tapenade, qui calcule analytiquement Yd. On compare les carrés scalaires des deux versions de Yd, qui doivent correspondre, modulo les perturbations du second ordre des différences divisées. Si le code tangent est valide, on initialise le code adjoint avec une valeur de Yb qui est la transposée de Yd. Le gradient rendu par le code adjoint est par définition X_b = Y_b × F'(X). Si le gradient est correct, il doit vérifier l'équation suivante:

$$Xb \times Xd = Yb \times F'(X) \times Xd = Yb \times Yd = Yd^2$$

Différences divisées:	Yd^2:	Xb × Xd :
0.36677744614799468081E+03	0.36656522267259947512E+03	0.36656522267259947512E+03

Figure 5: Résultats de la validation de l'adjoint produit par TAPENADE, par le test du produit scalaire.

Les performances de l'adjoint produit par TAPENADE sont en cours d'amélioration. Le schéma de checkpointing employé est le schéma par défaut (Checkpoint de chaque appel de sous-programme), qui conduit a sauver de très nombreux snapshots, y compris pour de très petites procédures, ce qui coûte cher en mémoire et en temps pour sauvegarder et restaurer ces snapshots. Nous expérimentons actuellement des schémas de checkpointing plus rationnels, en utilisant la directive « NOCHECKPOINT » de TAPENADE, et les performances semblent nettement s'améliorer. Les résultats avec le schéma par défaut sont un temps d'exécution moyen de 657 secondes, soit un facteur de ralentissement de 7.1. La consommation mémoire maximale due aux stockages est de 580 Mo, soit 6.7 fois l'occupation mémoire du code direct.

L'outil TAPENADE, utilisable librement à des fins de recherche, peut être utilisé de deux manières: soit comme un serveur web accessible à l'adresse http://tapenade.inria.fr:8080/tapenade, soit comme un exécutable installé sur la machine locale depuis le site ftp://ftp-sop.inria.fr/tropics/tapenade/. La documentation est disponible sur le site de l'équipe « Tropics »: http://www-sop.inria.fr/tropics/.

5 CONCLUSION

Nous avons obtenu par Différentiation Automatique un adjoint valide d'une configuration du code d'océanographie OPA 9.0 écrit en FORTRAN 95. Cet adjoint présente d'ores et déjà des performances comparables à celles de l'adjoint manuel d'OPA 8.0. Nous avons utilisé pour cela l'outil TAPENADE, et nous comptons poursuivre ce travail dans deux directions. D'une part nous allons construire l'adjoint de configurations plus réalistes, et d'autre part nous allons développer les fonctionnalités de TAPENADE vers des analyses plus fines et une meilleure interaction avec l'utilisateur numéricien, pour tendre vers la qualité des adjoints manuels. Notre objectif à moyen terme est d'automatiser au maximum, grâce à la DA, le processus de construction des codes adjoints pour l'assimilation variationnelle.

Références

[1] B. Barnier et al, *Impact of partial steps and momentum advection schemes in a the global ocean circulation model at eddy permitting resolution*, Ocean Dynamics (accepted), http://www.lodyc.jussieu.fr/NEMO/, 2006.

[2] H. M. Buecker, G. Corliss, P. Hovland, U. Naumann, B. Norris (eds.), *Automatic Differentiation: Applications, Theory, and Tools*, Springer, Lecture Notes in Computational Science and Engineering, 2005.

[3] A. Carle, M. Fagan, *ADIFOR 3.0 Overview*, Rice University report CAAM-TR-00-02, 2000.

[4] G. Corliss, C. Faure, A. Griewank, L. Hascoët, U. Naumann (eds.), *Automatic Differentiation of Algorithms: From Simulation to Optimization*, Springer, Computer and Information Science, 2001.

[5] F. Courty, A. Dervieux, B. Koobus, L. Hascoët, *Reverse Automatic Differentiation for Optimum Design: from adjoint state assembly to gradient computation*, Optim. Meth. & Softw., Vol.18, No. 5, pp. 615-627, 2003.

[6] R. Giering, *Tangent linear and Adjoint Model Compiler, Users manual,* http://www.autodiff.com/tamc, FastOpt Gmbh, 1997.

[7] A. Griewank, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM Frontiers in Applied Mathematics, 2000.

[8] L. Hascoët, U. Naumann, V. Pascual, *«To Be Recorded» analysis in reverse-mode Automatic Differentiation*, Elsevier, Future Generation Computer Systems, Vol. 21, No. 8, 2004.

[9] L. Hascoët, V. Pascual, *TAPENADE 2.1 User's Guide*, INRIA, Rapport technique 0300, http://www.inria.fr/rrrt/rt-0300.html, 2004.