INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet TROPICS

Transformations et Outils Informatiques pour le Calcul Scientifique

Sophia Antipolis

THÈME 1A

Table des matières

1 Composition de l'équipe

Responsable scientifique
Laurent Hascoët [CR]
Responsable permanent
Valérie Pascual [CR]
Assistante de projet
Nathalie Balax-Bellesso [TR, à temps partiel dans le projet]
Personnel Inria
Alain Dervieux [DR, depuis le 1/2/2001]
Rose-Marie Greborio [Ingénieur spécialiste industriel, depuis le 15/8/2001]
Chercheurs doctorants
David Leservoisier [CIFRE Snecma, jusqu'au 30/6/2001]
François Courty [Boursier INRIA, depuis le 1/1/2001]
Collaborateur extérieur
Bruno Koobus [Conseiller scientifique, Université de Montpellier]
Stagiaires
Fabien Dolla [Elève ESSI Sophia-Antipolis, du 15/6/2001 au 31/8/2001]
Marc-Aurèle Ngoxuan [Elève ESSI Sophia-Antipolis, du 15/6/2001 au 31/8/2001]
Jean-Charles Rihaoui [Elève ESSI Sophia-Antipolis, du 15/6/2001 au 31/8/2001]

2 Présentation et objectifs généraux

L'activité du projet tropics concerne les outils logiciels pour l'analyse et la transformation semi-automatique des programmes. Le domaine d'application visé est le calcul scientifique. Les objectifs principaux sont la Différentiation Automatique et la Parallélisation. Ces objectifs posent chacun des problèmes spécifiques, que nous devons étudier. Ces recherches se traduisent par des développements dans les outils maintenus par le projet, ODYSSÉE pour la différentiation, Partita pour la parallélisation. À terme, ces outils fusionneront pour donner une plate-forme commune d'analyse et de transformation de programmes scientifiques. Le projet est a priori candidat pour étudier et développer d'autres outils qui faciliteraient la tâche des développeurs en calcul scientifique.

Les axes de recherche du projet sont :

- La Différentiation Automatique : optimisations mémoire pour le mode inverse (adjoints) et les calculs de Jacobiennes, différentiation de programmes parallèles, différentiation adaptée de sous-programmes particuliers,
- La Parallélisation SPMD (Single Program, Multiple Data), appliquée aux programmes itératifs sur maillages irréguliers : détection et minimisation des besoins de communication entre processeurs,
- La comparaison de programmes, capable de détecter une équivalence plus sémantique que syntaxique.
- L'outillage commun aux logiciels de transformation de programmes scientifiques : représentation interne de gros programmes en langages impératifs, graphes d'appel, graphes de flot, graphes de dépendances.

Nous maintenons des relations avec les universités de Dresde (Allemagne), Aachen (Allemagne) et Hatfield (GB). Odyssée est distribué librement sur notre site FTP `ftp://ftp-sop.inria.fr/tropics`, et Partita est commercialisé par simulog.

3 Fondements scientifiques

3.1 Différentiation Automatique

Mots clés : transformation de programme, différentiation automatique, calcul numérique, simulation, optimisation, modèle adjoint.

Participants : François Courty, Fabien Dolla, Rose-Marie Greborio, Laurent Hascoët, Marc-Aurèle Ngoxuan, Valérie Pascual, Jean-Charles Rihaoui.

Glossaire :

différentiation automatique Transformation semi-automatique d'un programme initial, générant un programme «différentié» qui calcule de manière analytique certaines dérivées des résultats du programme initial par rapport à ses entrées.

modèle adjoint Manipulation des équations différentielles définissant un problème, pour obtenir des équations différentielles supplémentaires qui définissent le gradient de la solution du problème.

checkpointing Technique utilisée dans le mode inverse de la D.A., qui permet de réduire la quantité de mémoire consommée par le programme différentié pour sauvegarder les valeurs intermédiaires. Le checkpointing consiste à sélectionner une portion de code dans laquelle aucune valeur intermédiaire n'est sauvegardée. Si une telle valeur est ensuite demandée, la portion de code sera exécutée une deuxième fois, avec sauvegarde cette fois. Il s'agit donc d'un compromis stockage/recalcul.

La Différentiation Automatique (D.A.), est une technique qui, à partir d'un programme P implémentant une fonction f, génère un programme P' qui calcule certaines dérivées de f. Dans cette technique, on modélise le code sous la forme d'une composition de fonctions élémentaires. Générer le code différentié revient alors à appliquer la règle de dérivation des fonctions composées. L'étude de la D.A. a une longue tradition à l'INRIA [GLM91].

Il existe à l'heure actuelle deux approches pour obtenir P' :

– Par surcharge des opérateurs arithmétiques (outils ADOL-C [GJU96], ADOGEN [Roc]). Cette approche permet des développements rapides,

[GLM91] J. Gilbert, G. LeVey, J. Masse, « La différentiation automatique de fonctions representées par des programmes », *rapport de recherche n° 1557*, Inria, 1991, `http://www.inria.fr/rrrt/rr-1557.html`.

[GJU96] A. Griewank, D. Juedes, J. Utke, « Adol-C: a package for the automatic differentiation of algorithms written in C/C++ », *ACM Transactions on Mathematical Software 22*, 1996, p. 131–167.

[Roc] M. Rochette, *Manuel d'utilisation du logiciel ADOGEN*, Novacité Alpha, B.P. 2131, Villeurbanne Cedex.

- Par transformation de source à source (outils ODYSSÉE [4], ADIFOR [BCK+98], TAMC [Gie97], PADRE2 [Kub96]) ; cette approche demande le développement d'outils complexes, mais permet une plus grande flexibilité.

On trouvera une liste complète des systèmes actuels de D.A. à l'adresse : `http://www-sop.inria.fr/tropics/Odyssee/odyssee.html`. Notre projet s'intéresse essentiellement à l'approche «transformation de source à source».

Aux utilisations variées des programmes dérivés répondent divers *modes de différentiation*. Par exemple, on peut vouloir calculer les dérivées premières ou des dérivées d'ordre supérieur. Ensuite, a-t-on besoin de certaines dérivées directionnelles ou de la matrice jacobienne entière ? Enfin, on a le choix entre une propagation *directe* ou *inverse*. Pour présenter rapidement ces modes et les questions associées, considérons un programme initial P, $e_i, i \in [1,n]$ ses entrées, $r_j, j \in [1,m]$ ses résultats, et $f : \{e_1, ..., e_n\} \longmapsto \{r_1, ..., r_m\}$ la fonction implémentée par P.

- Le *mode «direct»* construit le programme

$$P' : \{e_1, ..., e_n, de_1, ..., de_n\} \longmapsto \{r_1, ..., r_m, dr_1, ..., dr_m\}$$

  qui calcule les variations dr_j des résultats de P en fonction des entrées e_i et de leurs variations de_i, c'est-à-dire une variation suivant une direction donnée de l'espace des entrées $\mathbb{R}^n$. P' calcule en fait la fonction linéaire «tangente» à f au point $\{e_1, ..., e_n\}$. Ce mode «fondamental» a été relativement bien étudié d'un point de vue théorique. On a une bonne estimation de sa complexité et de sa consommation mémoire [Iri91]. Divers raffinements intéressants ont été proposés, tels que le calcul simultané pour plusieurs directions de variation des entrées.
- Le *mode «matrice jacobienne»* construit le programme

$$P' : \{e_1, ..., e_n\} \longmapsto \{r_1, ..., r_m, \frac{\partial r_1}{\partial e_1}, ..., \frac{\partial r_j}{\partial e_i}, ..., \frac{\partial r_m}{\partial e_n}\}$$

  qui calcule la matrice jacobienne des résultats par rapport aux entrées. La difficulté principale réside dans la taille de la matrice jacobienne $n * m$, ainsi que la taille de toutes les jacobiennes intermédiaires durant l'évaluation de P'. Cela peut rendre catastrophiques les performances temps et mémoire de P'. Pour répondre à ce problème, on doit utiliser le fait que la Jacobienne est une matrice probablement creuse.
- Le *mode «inverse»* construit le programme

$$P' : \{e_1, ..., e_n, dr_1^*, ..., dr_m^*\} \longmapsto \{r_1, ..., r_m, de_1^*, ..., de_n^*\}$$

[BCK+98] C. Bischof, A. Carle, P. Khademi, A. Mauer, P. Hovland, « ADIFOR2.0 User's Guide », *rapport de recherche*, Argonne National Laboratory Technical Memorandum ANL/MCS-TM-192, and CRPC Technical Report CRPC-TR95516-S, 1998.

[Gie97] R. Giering, *Tangent linear and Adjoint Model Compiler , Users manual 1.2*, 1997, `http://puddle.mit.edu/~ralf/tamc`.

[Kub96] K. Kubota, *PADRE2 version 2α, user's manual*, 1996.

[Iri91] M. Iri, « History of automatic differentiation and rounding estimation », *in : Automatic Differentiation of Algorithms: Theory, Implementation and Application*, A. Griewank, G. Corliss (éditeurs), SIAM, p. 1–16, 1991.

Ce nouveau programme prend en paramètres supplémentaires des valeurs «adjointes» dr_j^*, que l'on peut comprendre comme des pondérations des résultats r_j. Ceci définit un critère d'optimisation $\sum_j dr_j^*.r_j$. Ce nouveau programme va calculer la direction de variation de_i^* des entrées, qui maximise la variation de ce critère d'optimisation. Ceci revient à dire que l'on calcule la direction de sensibilité maximale de f, ou encore son gradient. Ce mode présente un avantage dans le cas où le programme P a un petit nombre de résultats ($m << n$), la complexité du mode direct étant liée au nombre d'entrées n, alors que celle du mode inverse est liée au nombre de résultats m [Iri91]. Le mode inverse est particulièrement intéressant parce qu'il correspond à une génération automatique du code «adjoint». La méthode dite de l'adjoint consiste à discrétiser un nouveau problème dont les inconnues sont les dérivées cherchées. Dans le cas où le modèle est mathématiquement adéquat, cette méthode donne de bons résultats. Elle est cependant difficile à mettre en œuvre, en particulier parce qu'elle introduit de nouvelles quantités «duales» qui n'ont aucune signification physique. D'où l'intérêt d'une génération automatique. Cependant, l'adjoint (et le mode inverse) utilise les valeurs intermédiaires calculées par P, dans l'ordre *inverse*. Ces valeurs doivent donc être stockées ou recalculées. Cela pose de graves problèmes de place mémoire, qui rendent le mode inverse dangereusement coûteux en l'état actuel de la technique. On étudie donc des tactiques pour limiter les mémorisations inutiles [CG97]. Les compromis stockage/recalcul font aussi l'objet de recherches actives [Gri92], [GPRS96], [Cha98].

Parallèlement aux problèmes spécifiques de chacun des modes précédents, on s'intéresse à des questions plus globales, telles que :

- Le lien avec la parallélisation. Comment conserver la parallélisation d'un programme lors de sa différentiation ? Comment différentier les diverses formes d'expression du parallélisme ? Comment utiliser la parallélisation pour des programmes différentiés plus efficaces ?
- La différentiation au voisinage des points singuliers des programmes, tels que les conditionnelles, qui induisent une discontinuité de la fonction représentée par le programme.
- L'étude fine de cas particuliers, pour lesquels il existe une différentiation spécifique plus adaptée. Citons les résolutions itératives, ou certaines opérations classiques d'algèbre linéaire.
- L'étude de l'interaction avec l'utilisateur. Il est toujours nécessaire de permettre à l'utilisateur de donner des informations complexes sur le programme, ou de désigner des fragments pour lesquels existe une méthode de différentiation spécialement efficace.

[CG97] I. Charpentier, M. Ghemires, « Génération automatique de codes adjoints : Stratégies d'utilisation pour Odyssée, application au code meso-nh », *rapport de recherche n° 3251*, Inria, 1997, `http://www.inria.fr/rrrt/rr-3251.html`.

[Gri92] A. Griewank, « Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation », *Optimization Methods and Software 1*, 1992, p. 35–54.

[GPRS96] J. Grimm, L. Pottier, N. Rostaing-Schmidt, « Optimal time and minimum space-time product for reversing a certain class of programs », *in : Computational Differentiation: Techniques, Applications and Tools*, M. Berz, C. Bischof, G. Corliss, A. Griewank (éditeurs), SIAM, p. 95–106, 1996. rapport de Recherche INRIA 2794, `http://www.inria.fr/rrrt/rr-2794.html`.

[Cha98] I. Charpentier, « Génération de codes adjoints : Traitement de la trajectoire du modèle direct », *rapport de recherche n° 3405*, Inria, 1998, `http://www.inria.fr/rrrt/rr-3405.html`.

3.2 Parallélisation

Mots clés : transformation de programme, parallélisation, optimisation de code, compilation, OpenMP, SPMD.

Participants : Laurent Hascoët, Fabien Dolla, Valérie Pascual.

Glossaire :
parallélisation Transformation semi-automatique de programme produisant un nouveau programme de même sémantique, mais exploitant au mieux les possibilités d'une combinaison donnée de *langage cible*, *compilateur*, et *architecture machine cible*.

SPMD «Single Program Multiple Data». Modèle d'exécution parallèle, dans lequel plusieurs copies d'un même programme s'exécutent en parallèle, sur des ensembles de données différents, les diverses exécutions communiquant entre elles par échange de messages.

A l'heure actuelle, l'activité autour de la Différentiation Automatique nous occupe beaucoup, et il reste peu de temps pour la parallélisation. Néanmoins cela reste un objectif à moyen terme. En prévision, nous continuons d'inclure dans notre base logicielle des algorithmes utiles à la parallélisation, tels que l'analyse in-out ou l'extraction de sous-programme.

Nous ne nous occupons pas ici de *programmation parallèle*, qui consiste à écrire une application en appliquant un style de programmation parallèle donné, mais plutôt de *parallélisation*, qui consiste à transformer, grâce à un outil, un programme existant pour l'adapter à un tel style parallèle.

Il faut faire une autre distinction entre la parallélisation des instructions (ou des opérations), qui recherche les instructions indépendantes, pour les exécuter en parallèle, et l'optimisation de la localité mémoire, qui recherche un placement des variables dans la mémoire et/ou un ordre des instructions qui minimise les communications et le trafic mémoire. Historiquement, la parallélisation des instructions a été le premier sujet abordé, à destination des architectures vectorielles. Les optimisations liées aux communications ont été étudiées plus récemment, mais occupent une place prépondérante, par l'importance des architectures parallèles à mémoire distribuée. Ces deux activités sont proches, et s'appuient toutes deux sur le calcul fin des *dépendances* entre les lectures et les écritures des valeurs des variables.

Il existe aussi une distinction de «grain de parallélisation». On peut rechercher une parallélisation à grain fin, entre les opérations atomiques du programme. Cette question est proche de la vectorisation. On peut au contraire considérer de larges fragments du programme comme atomiques, et rechercher des exécutions concurrentes entre ces fragments. Ce type de parallélisation convient plus aux architectures parallèles à mémoire distribuée. Notre activité de cette année s'inscrit plutôt dans cette deuxième direction.

Pour isoler des processus indépendants à l'intérieur d'un programme donné, il est indispensable d'extraire des fragments de programme. Cette extraction demande d'une part de vérifier que le fragment considéré constituerait un sous-programme valide pour le contrôle, c'est-à-dire comportant un point d'entrée et un point de sortie unique. Elle demande d'autre part de déterminer les ensembles minima de variables (arguments ou globales) qui doivent être passés en paramètre, par valeur ou par référence. Cela donne une bonne mesure des dépendances entre les futurs processus.

L'aide à la parallélisation SPMD est davantage guidée par les demandes des utilisateurs nu-

mériciens. On constate que le grand nombre d'approches différentes de la parallélisation permet aux utilisateurs finaux de définir leur propre modèle de programmation parallèle, plus adapté aux caractéristiques de leurs applications. Chacun de ces modèles nécessite des outils d'aide spécifiques. On part ainsi d'une méthode de parallélisation plutôt empirique, que l'on doit formaliser et généraliser, pour aboutir à la spécification d'un outil d'aide. Ces considérations générales s'appliquent parfaitement à la parallélisation SPMD. On est parti d'une méthode de parallélisation manuelle existante, plutôt élégante et efficace, développée dans le projet SINUS. On a ensuite spécifié un outil, et développé un prototype [6] [5] [9], pour le placement optimal des appels aux routines de communication. On cherche à améliorer cet outil en étudiant son utilisation sur des programmes réels.

3.3 Analyses et transformations de programmes

Mots clés : analyse de programme, transformation de programme, compilation, arbre de syntaxe abstraite, graphe de flot de contrôle, interprétation abstraite, graphe de dépendances.

Participants : Fabien Dolla, Rose-Marie Greborio, Laurent Hascoët, Marc-Aurèle Ngoxuan, Valérie Pascual, Jean-Charles Rihaoui.

Glossaire :

arbres de syntaxe abstraite Représentation arborescente d'un sous-programme, dans laquelle seules sont conservées les informations définissant le sens, la sémantique, du sous-programme, et dans laquelle les traits de syntaxe (indentation, parenthésage,...) ont été abstraits.

graphe de flot de contrôle Représentation du corps d'un sous-programme sous forme d'un graphe, dont les nœuds sont des listes d'instructions en séquence, et dont les flèches représentent les sauts du contrôle lors des tests, des boucles,...

interprétation abstraite Modèle abstrait de description des analyses de programmes, dans lequel on simule une exécution, où les valeurs des variables sont remplacées par des valeurs abstraites dans un certain *domaine sémantique*. Pour chaque analyse, on définit un domaine sémantique particulier.

graphe de dépendances Graphe reliant les accès en lecture et écriture des variables d'un sous-programme. Les dépendances relient soit une écriture d'une valeur vers une lecture de la même valeur, soit une lecture (ou une écriture) d'une valeur vers une autre écriture qui peut écraser cette valeur. Les dépendances expriment une relation d'ordre nécessaire entre les opérations.

L'analyse et la transformation de programmes sont des activités anciennes et classiques. Ce sont entre autres les opérations essentielles des compilateurs [ASU86]. Après les outils de manipulation spécialisés pour un langage particulier sont apparus des environnements aidant à spécifier de tels outils pour le langage de son choix. Nous rangeons dans cette catégorie le

[ASU86] A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986.

Cornell Synthesizer [RT89] et CENTAUR [BCD+88,Inr94]. Dans ces environnements, comme dans les compilateurs récents, on trouve un découpage plus net entre le «front end» (analyseur syntaxique), les analyses et transformations proprement dites, et le «back end» (afficheur ou générateur de code). Ce découpage nécessite une représentation interne normalisée des programmes, sous forme d'arbres syntaxiques (AST). Il apparaît aussi, dans les compilateurs, l'idée d'une forme interne (ou intermédiaire) commune à plusieurs langages d'origine, s'ils sont raisonnablement proches. L'AST est un bon support pour cette forme intermédiaire.

Alors que les AST sont la seule représentation des programmes acceptée par les environnements génériques, une autre représentation, les graphes de flot de contrôle (FG), est utile pour les analyses complexes et les optimisations, ainsi que dans les outils spécialisés existants (ODYSSÉE ou PARTITA). En effet, l'expérience de CENTAUR, par exemple, montre que l'usage exclusif des AST handicape les outils des environnements génériques, en limitant les analyses et optimisations globales, et en traitant mal les instructions de contrôle non structurées.

Trop d'analyses sont fondées sur des graphes pour se contenter d'une représentation d'arbres. En contrepartie, les analyses et transformations sur les arbres [Kah87,APR97], peuvent être spécifiées d'une manière suffisamment abstraite (sémantique naturelle, grammaires attribuées) pour envisager des preuves de correction. En revanche, les FG sont un support naturel pour des analyses et interprètes de programmes quelconques, non structurés, en particulier par des méthodes d'interprétation abstraite [Cou96], ou pour des optimisations globales (code mort, variables vivantes, code invariant...).

Pour favoriser l'utilisation de ces graphes, on se propose de développer une plate-forme commune, qui fournirait ces graphes et leurs analyses associées, pour les langages impératifs en général. On cherche à définir une API permettant à un outil particulier (par exemple de D.A) de s'appuyer sur cette plate-forme.

L'autre outil essentiel est le *graphe de dépendances*. Suivant l'utilisation (parallélisation, différentiation, comparaison de programmes...), on en utilise diverses variantes, dont le *data-flow graph*, le *data-dependence graph* [Kuc78] [AK87], le *program dependence graph* [FOW87] ou le *dependence flow graph* [PBJ+91]. L'utilisation de ces graphes dans le cas particulier de la

[RT89] T. REPS, T. TEITELBAUM, *The Synthesizer Generator Reference Manual*, Springer-Verlag, 1989.

[BCD+88] P. BORRAS, D. CLEMENT, T. DESPEYROUX, J. INCERPI, G. KAHN, B. LANG, V. PASCUAL, « Centaur: the system », *Proceedings of ACM SIGSOFT'88: Third Symposium on Software Development Environments, Boston*, November 1988, (aussi Inria rapport de Recherche No 777).

[Inr94] INRIA, *Centaur 2.0 Documentation*, 1994.

[Kah87] G. KAHN, « Natural Semantics », *Lecture Notes in Computer Science 247*, 1987, Proceedings of STACS 1987.

[APR97] I. ATTALI, V. PASCUAL, C. ROUDET, « A language and an integrated environment for program transformations », *rapport de recherche n° 3313*, Inria, 1997, `http://www.inria.fr/RRRT/RR-3313.html`.

[Cou96] P. COUSOT, « Abstract Interpretation », *ACM Computing Surveys 28*, 1, 1996, p. 324–328.

[Kuc78] D. KUCK, *The structure of computers and computations*, *1*, Wiley, 1978.

[AK87] J. ALLEN, K. KENNEDY, « Automatic translation of Fortran programs to vector form », *ACM Transactions on Programming Languages and Systems 9*, 4, 1987, p. 491–542.

[FOW87] J. FERRANTE, K. OTTENSTEIN, J. WARREN, « The Program Dependence Graph and its use in optimization », *ACM Transactions on Programming Languages and Systems 9*, 3, 1987, p. 319–349.

[PBJ+91] K. PINGALI, M. BECK, R. JOHNSON, M. MOUDGILL, R. STODGHILL, *Dependence Flow Graphs:*

parallélisation est décrit dans [ZC90], [Fea89], [AK87] ou [Dar93] et, pour la comparaison de sous-programmes, dans [RHY89,Ang96]. Il serait intéressant d'unifier ces variantes au sein de la plate-forme d'analyse de programmes impératifs.

Pour que ce travail soit réellement utile, il est très important de s'abstraire dès le début d'un langage impératif particulier, tel que FORTRAN77. Dans ce but, on a défini un langage impératif abstrait, vers lequel on peut projeter chaque langage impératif réel. La plate-forme ne connaît que ce langage abstrait, ce qui facilite grandement le passage d'un langage à l'autre.

3.4 Applications à la Mécanique des Fluides Numérique

Mots clés : Optimisation, gradient, écoulements compressibles.

Participants : François Courty, Alain Dervieux, Laurent Hascoët, Bruno Koobus.

Glossaire :

linéarisation La modélisation des phénomènes de la Mécanique des Milieux Continus aboutit à des Equations aux Dérivées partielles. La linéarisation permet de modéliser le comportement de petites perturbations, soit parce qu'elles sont effectivement petites (c'est le cas pour les ondes acoustiques), soit parce qu'on cherche à évaluer la sensibilité d'un système par rapport à un paramètre, lequel sera éventuellement touché par des modifications non nécessairement petites (c'est le cas en optimisation).

état adjoint Lorsqu'on s'intéresse à une caractéristique scalaire d'un système, sa sensibilité aux paramètres pourra s'exprimer par la solution d'une équation issue de l'équation initale par linéarisation et tranposition et dont la solution est l'état adjoint.

Une des propriétés originales de l'équipe Tropics est sa configuration multi-disciplinaire. Cette configuration sans doute provisoire a pour but la mise en œuvre par une partie de l'équipe d'applications démonstratives de haute complexité et concernera *l'optimisation de forme en l'aérodynamique.*

Les outils de Différentiation Automatique développés et maintenus par Tropics sont appliqués dans ce contexte de facon (1) à résoudre de difficiles problèmes annexes, (2) à fournir aux développeurs de ALI des informations sur le fonctionnement de l'existant, et (3) à contribuer à la spécification des nouveaux outils. Les problèmes annexes à résoudre sont des problèmes ouverts en Mécanique des Fluides Numérique tels que l'approximation ou la résolution d'un système adjoint pour un écoulement compressible ainsi que des problèmes en optimisation

---

an algebraic approach to program dependencies, Pitman, 1991.

[ZC90] H. Zima, B. Chapman, *Supercompilers for Parallel and Vector Computers*, ACM Press, 1990.

[Fea89] P. Feautrier, « Semantical analysis and mathematical programming; application to parallelization and vectorization », *in : Workshop on Parallel and Distributed Algorithms*, M. Cosnard, Y. Robert, P. Quinton, M. Raynal (éditeurs), North Holland, p. 309–320, Bonas, 1989.

[Dar93] A. Darte, *Techniques de parallélisation automatique de nids de boucles*, thèse de doctorat, ENS Lyon, université Lyon-1, 1993.

[RHY89] T. Reps, S. Horwitz, W. Yang, « Detecting program components with equivalent behaviors », *rapport de recherche n° 840*, Computer Sciences Lab., University of Wisconsin, Madison, 1989.

[Ang96] L. Angeli, *Factorisation de sous-programmes*, thèse de doctorat, université de Nice Sophia-Antipolis, 1996.

tels que la mise au point d'algorithmes de minimisation avec contraintes utilisant de manière optimale les outils fournis par la Différentiation Automatique.

4 Domaines d'applications

4.1 Panorama

Le domaine d'application du projet concerne les programmes de calcul scientifique, écrits dans un langage impératif classique, tel que FORTRAN, FORTRAN90, C, C++,... Notre but est de fournir un ensemble d'outils d'aide pour l'analyse et la transformation de ces programmes. Notre application phare est la Différentiation Automatique, suivie de la parallélisation. D'une manière générale, les dérivées d'un programme sont utiles pour plusieurs types d'applications :

- la simulation de systèmes complexes,
- les études de sensibilité,
- l'analyse des propagations d'erreurs d'arrondi,
- la mise en œuvre de l'itération de Newton,
- l'intégration implicite de problèmes d'évolution (équations différentielles ordinaires ou équations aux dérivées partielles), en particulier pour les problèmes raides.
- les estimations d'erreur d'approximation,
- les problèmes inverses, tels que l'assimilation des données [LT86] [Tal91],
- les méthodes d'optimisation de formes, les méthodes d'optimisation sous contraintes, et plus généralement tous les algorithmes basés sur une linéarisation locale,

Dans ce qui suit, nous allons détailler certaines de ces applications de la Différentiation Automatique.

4.2 Simulation

Pour simuler par programme un système complexe, on est souvent amené à résoudre des systèmes d'équations différentielles. Ces équations modélisent le système réel que l'on veut simuler. La résolution de ces équations par un programme est coûteuse et difficilement compatible avec des contraintes du type temps réel. Lorsque l'on veut simuler la variation de l'état d'un système en réponse à de petites perturbations des paramètres, il existe une solution approchée efficace : on suppose que le comportement du système est linéaire dans un petit voisinage du point initial. C'est par exemple la démarche fondamentale conduisant aux équations de propagation du son. La modification de l'état en réponse à une modification des paramètres peut être calculée comme un simple produit matrice-vecteur. La matrice est constante si on reste dans le même voisinage de l'état initial. Cette matrice (matrice *jacobienne*) peut être calculée grâce à la Différentiation Automatique du programme initial.

[LT86] F. LeDimet, O. Talagrand, « Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects », *Tellus 38A*, 1986, p. 97–110.

[Tal91] O. Talagrand, « The use of adjoint equations in numerical modelling of the atmospheric circulation », *in: Automatic Differentiation of Algorithms: Theory, Implementation and Application*, A. Griewank, G. Corliss (éditeurs), SIAM, p. 169–180, 1991.

4.3 Assimilation de données

On considère les problèmes de prévisions météorologiques, ou les questions associées liées à l'environnement, comme la prévision des changements climatiques. Dans ces problèmes, les erreurs dans la détermination de l'état initial sont une cause majeure des erreurs sur la prévision. Ces erreurs sont souvent prépondérantes par rapport à celles provenant de l'approximation dans la modélisation du comportement. Les données initiales sont souvent collectées en des lieux imprécis, à des moments différents et imprécis également. Comment déterminer le meilleur état initial, c'est-à-dire le plus proche des données mesurées ? La réponse passe par une minimisation du type «moindres carrés», qui fait appel au calcul de l'état adjoint. Cet adjoint peut être calculé par un programme écrit à la main, mais il peut aussi être calculé grâce au mode inverse de la Différentiation Automatique. Cette approche a déjà été validée, par exemple dans la thèse de Nicole Rostaing.

4.4 Optimisation de formes en mécanique des fluides

Un programme de calcul d'écoulement en mécanique des fluides est capable de calculer la valeur d'un certain nombre de critères, par exemple la portance, en fonction de paramètres initiaux, par exemple la géométrie d'une aile d'avion. Lorsque l'on veut trouver la valeur de ces paramètres qui optimise le critère (ou une certaine combinaison des critères), on peut employer une méthode de descente par gradient. On a donc besoin du gradient du critère par rapport aux paramètres. Ce gradient peut être obtenu de diverses manières, parmi lesquelles la Différentiation Automatique fournit les résultats les plus précis. En particulier, la Différentiation Automatique en mode inverse proposée par ODYSSÉE est une approche très prometteuse, qui a déjà fait l'objet de publications [MRM96,HMB97].

5 Logiciels

5.1 ALI

Participants : Laurent Hascoët [correspondant], Fabien Dolla, Rose-Marie Greborio, Marc-Aurèle Ngoxuan, Valérie Pascual, Jean-Charles Rihaoui.

ALI est une plate-forme pour l'*Analyse des Langages Impératifs*, répondant aux objectifs de la section 3.3. ALI est destiné à servir de base commune aux outils développés par le projet. En particulier, le nouvel outil de Différentiation Automatique du projet, TAPENADE, est bâti au-dessus d'ALI. Jusqu'à cette année, il faisait encore appel à ODYSSÉE. Cette année, nous avons développé une nouvelle différentiation en modes direct et inverse, totalement découplée d'ODYSSÉE.

ALI accepte en entrée des programmes décrits dans une syntaxe abstraite particulière, nommée IL, indépendante de tel ou tel langage impératif classique. Seule la syntaxe *abstraite* d'IL

[MRM96] J. MALE, N. ROSTAING, N. MARCO, « Automatic Differentiation: an application to Optimum Shape Design in Aeronautics », Wiley, 1996. Proceedings of ECCOMAS'96.

[HMB97] P. HOVLAND, B. MOHAMMADI, C. BISCHOF, « Automatic Differentiation and Navier-Stokes Computations », *rapport de recherche n° ANL/MCS-P687-0997*, Argonne National Laboratory, 1997.

est définie, et elle recouvre les constructeurs syntaxiques des principaux langages impératifs, tels que FORTRAN, FORTRAN90, C, C++... ALI peut donc s'interfacer avec un langage impératif quelconque. Cette année, nous avons développé un parseur et un afficheur pour FORTRAN77 et FORTRAN90. Il n'existe encore qu'un parseur pour C. De manière standard, ALI construit une représentation interne d'un haut niveau d'abstraction, composée d'un ou plusieurs graphes d'appel, de graphes de flot de contrôle et de tables de symboles, et effectue systématiquement des analyses statiques d'intérêt général (type-checking, ...) facilitant le travail d'analyses spécifiques ultérieures. Une première ébauche d'API (Application Programmer Interface) a été définie, pour permettre à un développeur d'application de construire des analyses et transformations spécifiques à partir des structures fournies par ALI. Il peut s'agir de Différentiation, ce qui est fait avec TAPENADE, mais également de parallélisation, évaluation partielle, et autres optimisations de source.

ALI est implémenté sous la forme d'un ensemble de classes JAVA, et utilise la représentation des arbres de syntaxe abstraite fournie par la bibliothèque AIOLI. Le développement d'ALI a démarré en 1999.

5.2 TAPENADE

Participants : Laurent Hascoët [correspondant], Fabien Dolla, Rose-Marie Greborio, Marc-Aurèle Ngoxuan, Valérie Pascual, Jean-Charles Rihaoui.

TAPENADE est le nouvel outil de D.A. proposé par le projet. Il reprend de nombreux algorithmes d'ODYSSÉE en les améliorant. TAPENADE est construit en Java au-dessus de la plateforme ALI. TAPENADE a vocation à remplacer ODYSSÉE à court terme, et tous les efforts de développement du projet lui sont désormais consacrés.

TAPENADE est un système de différentiation automatique par transformation de programme source Fortran (77 et 90). Etant donnés une partie du graphe d'appel, un ensemble de variables de sortie (dites «dépendantes») dont on demande les dérivées, et un ensemble de variables d'entrée (dites «indépendantes») par rapport auxquelles on veut ces dérivées, TAPENADE construit un nouveau programme source qui calcule ces dérivées, soit en mode direct, ce qui donne les sensibilités, soit en mode inverse, ce qui donne le gradient.

Le mode inverse de TAPENADE implémente un modèle de différentiation innovant, entièrement fondé sur le graphe de flot de contrôle, décrit dans la partie résultats (6.2).

TAPENADE met en œuvre un certain nombre de techniques et d'analyses pour produire un résultat efficace. Outre le calcul des dépendances et la détection des variables actives, déjà validées dans ODYSSÉE mais grandement améliorées ici, il implémente des techniques nouvelles parmi lesquelles :

- L'analyse d'*utilité*, réciproque de l'activité, qui permet de ne pas différentier les variables qui n'ont pas d'influence sur les sorties dépendantes spécifiées par l'utilisateur. Cela est mieux décrit dans la section 6.3.
- L'analyse de sauvegarde en mode inverse, qui permet de limiter les sauvegardes aux valeur effectivement utilisées dans les dérivées partielles durant la phase inverse. Cette analyse avait été expérimentée par Uwe Naumann dans ODYSSÉE.
- Le calcul du graphe de dépendances local à chaque bloc d'instructions différentiées. Cela

permet de modifier l'ordre de ces instructions, et partant de les regrouper pour obtenir un code plus concis qui améliore la localité des accés mémoire. Cela est décrit dans la section 6.2.
– L'analyse In-Out, classique en compilation, est utilisée pour réduire la taille mémoire nécessaire à l'exécution répétée de certains sous-programmes. Ceci augmente l'efficacité du «checkpointing».

TAPENADE s'utilise soit depuis la ligne de commande, comme un compilateur, soit via une interface graphique XHTML. Une version expérimentale «alpha» de TAPENADE est directement accessible et utilisable à travers le web, à partir du site du projet `http://www-sop.inria.fr/tropics/`. Il ne s'agit pas d'une page de téléchargement de l'outil, mais d'une servlet permettant une utilisation *directe* de TAPENADE. Cela est mieux décrit dans la section 6.5.

5.3 Odyssée

Participants : Laurent Hascoët [correspondant], Valérie Pascual.

Mots clés : différentiation automatique, transformation de code, code adjoint, dérivée.

ODYSSÉE est un système de différentiation automatique qui fonctionne par transformation de programme Fortran 77. Il est écrit en Caml. ODYSSÉE génère un code différentié soit en mode direct, soit en mode inverse. La version 1.7 d'ODYSSÉE Odyssée est en accès libre, avec le source Caml, sur le serveur FTP du projet `ftp://ftp-sop.inria.fr/tropics`. Tous les efforts de développement du projet portent désormais sur le *successeur* d'ODYSSÉE, TAPENADE (5.2), qui est bâti au-dessus de la plateforme ALI (5.1). ODYSSÉE reste accessible et utilisable, mais il ne fait plus l'objet de développements nouveaux.

5.4 Icare

Participants : François Courty [correspondant], Alain Dervieux, Bruno Koobus.

Mots clés : Optimisation de forme, adjoint, écoulement, compressible.

ICARE est un logiciel (Fortran) permettant d'optimiser par une méthode de gradient une fonctionnelle dépendant d'une fonction d'état aérodynamique. Le modèle aérodynamique est celui des équations d'Euler, en 2 dimensions. Le paramètre à optimiser est la forme de l'objet immergé dans l'écoulement. La chaine complète de sensibilité reposant sur un calcul d'adjoint discret est issue de la différentiation par ODYSSÉE de noyaux fluides originaux.

6 Résultats nouveaux

6.1 Isomorphisme du Graphe de Dépendances en mode inverse

Mots clés : différentiation automatique, mode inverse, graphe de dépendances.

Participant : Laurent Hascoët.

La Différentiation Automatique en mode inverse génère ce que nous appelons un programme

adjoint. Ce long programme résulte d'une transformation complexe, et il est non seulement difficile à lire par un humain, mais aussi difficile à analyser ou optimiser par un outil informatique. D'autre part, de nombreux auteurs ont proposé des optimisations sur ce mode inverse. Mais ces optimisations, souvent astucieuses, sont présentées de manière informelle, ce qui empêche toute validation ou justification formelle. Les outils qui proposent ces différentiations optimisées ne peuvent donc garantir leur validité.

Dans la majorité des cas, ces optimisations reposent sur des modifications locales du programme adjoint standard, telles que des modifications du flot de contrôle, des transformations de boucles, ou des fusions, coupures ou échanges d'instructions. Il est possible de justifier ces optimisations en utilisant la notion de graphe de dépendances. Cette notion est utilisée intensivement en parallélisation par exemple.

Dans ce but, nous avons étudié le graphe de dépendances des programmes adjoints. Nous avons remarqué puis prouvé un lien très fort entre le graphe de dépendances du programme adjoint et celui du programme initial non différentié. Moyennant certaines hypothèses précises, dont l'*atomicité* des opérations d'incrémentation, le graphe de l'adjoint est isomorphe à un sous-graphe du graphe initial. La démonstration complète de cette propriété est donnée en [12].

Ce résultat permet de valider la méthode «iteration-wise adjoining» proposée en 2000, ainsi que d'autres améliorations du mode inverse de la D.A., reposant sur des changements de l'ordre des instructions. Il montre aussi dans quelle mesure les propriétés de parallélisme d'un code sont conservées par la différentiation en mode inverse.

Nous envisageons d'utiliser cet isomorphisme pour valider d'autres aspects de la différentiation en mode inverse, tels que la sauvegarde sélective des valeurs intermédiaires et les compromis stockage/recalcul.

6.2 Différentiation en mode inverse sur le Graphe de Contrôle

Mots clés : différentiation automatique, mode inverse, graphe de contrôle.

Participant : Laurent Hascoët.

Dans le cadre du développement de TAPENADE, le successeur d'ODYSSÉE, nous avons défini et implémenté une nouvelle approche de l'inversion du contrôle pour le mode inverse de la D.A.

Rappelons tout d'abord l'approche précédente, adoptée par ODYSSÉE. Un programme différentié en mode inverse comporte essentiellement deux phases : une phase «directe», grosso modo identique au programme de départ, suivie d'une phase, dite «inverse», pendant laquelle sont effectuées les instructions adjointes de celles effectuées pendant la phase directe, dans l'ordre exactement *inverse*. La question est «comment inverser le contrôle d'un programme ?». La présence d'instructions de saut, de tests imbriqués ou de sorties de boucles, fait que le contrôle inverse n'est pas nécessairement aussi bien structuré que le contrôle direct. Dans ODYSSÉE, le contrôle inverse est tout simplement mis à plat. On mémorise dynamiquement, durant la phase directe, la longue liste des blocs d'instructions effectués, puis on parcourt cette liste à l'envers pour contrôler la phase inverse. Cette approche présente certains inconvénients. Outre le fait que le programme inverse devient illisible humainement, il devient réfractaire à

toute forme d'optimisation par le compilateur. En effet, le contrôle est devenu dynamique. Toute notion de boucle ou de structure a disparu.

Nous proposons une approche différente, basée sur le graphe de flot de contrôle, qui est une représentation classique en compilation. L'inversion du contrôle est effectuée sur ce graphe. Ainsi les boucles sont conservées, de même que les instructions conditionnelles. Le programme inverse conserve l'essentiel de la structure de contrôle. La mémorisation du contrôle, toujours nécessaire, occupe beaucoup moins de mémoire. Nous avons implémenté cette nouvelle approche dans l'outil TAPENADE.

Nous avons aussi étudié le regroupement des instructions adjointes. Une nouvelle unité de différentiation se dégage : le bloc d'instructions. Nous proposons une nouvelle optimisation du mode inverse : la fusion des instructions adjointes dans un bloc d'instructions. En mode inverse, chaque instruction d'origine peut engendrer plusieurs instructions adjointes. La partie adjointe est alors très longue et complexe. Un compilateur a de grandes chances de ne rien optimiser. En particulier, il ne pourra pas fusionner certaines instructions adjointes parce qu'elles sont séparées par une distance plus grande que la fenêtre d'optimisation («peep-hole optimization»). Or, ces instructions ont été générées par la D.A. Nous savons que les dépendances de données entre elles sont isomorphes aux dépendances entre les instructions originelles [12]. Pour un bloc donné, nous construisons le graphe de dépendances, grâce auquel nous pouvons étudier les regroupements d'instructions. Par exemple, nous regroupons ainsi deux incréments d'une même variable, ou deux initialisations, ou une initialisation suivie d'un incrément. Le nouveau différentiateur nous permettra de mesurer l'impact de cette optimisation.

6.3 Différentiation sélective selon les résultats

Mots clés : différentiation automatique, slicing.

Participants : Laurent Hascoët, Marc-Aurèle Ngoxuan.

ODYSSEE, comme la plupart des outils de D.A. permet de désigner les variables d'entrée par rapport auxquelles on veut différentier. On les appelle les variables *indépendantes*. Nous avons introduit dans TAPENADE la notion duale. Nous voulons désigner les variables de sortie dont on demande la dérivée. Ces variables sont classiquement appelées *dépendantes*.

Classiquement, on propage les variables *indépendantes* dans le programme, par une analyse statique. Les variables qui en dépendent sont appelées *actives*. A l'intérieur du programme, les variables non actives n'auront pas besoin d'être dérivées (puisque leur dérivée est certainement nulle). De même, en propageant les variables *dépendantes* dans le programme, par une analyse statique cette fois dans l'ordre inverse de l'ordre d'exécution, on décide que de nouvelles variables n'auront pas besoin d'être dérivées, car leur dérivée n'est utile à aucune dérivée utile en sortie.

Nous avons défini et implémenté cette analyse. Elle est similaire au classique *slicing*. Il s'agit en fait d'un slicing sur le programme différentié, avant même la géneration de ce dernier. Il reste à mesurer sur des exemples réels le bénéfice apporté par cette analyse.

6.4 Extraction de Fragments de Programme

Mots clés : différentiation automatique, parallélisation, transformation de programmes.

Participants : Fabien Dolla, Laurent Hascoët, Valérie Pascual.

Toutes les transformations de programmes, et spécialement les transformations complexes, nécessitent de manipuler des fragments de programmes. L'une des raisons principales est qu'une transformation complexe est le plus souvent semi-automatique : l'utilisateur doit donner des directives à l'outil, et ces directives s'appliquent rarement à un sous-programme entier. Par exemple, en parallélisation par extraction de tâches, on doit laisser l'utilisateur désigner interactivement les tâches, qui sont des fragments du programme. En différentiation, on ne différentie pas la totalité d'un programme, mais seulement un fragment. En mode inverse, c'est en dernier ressort l'utilisateur qui doit désigner les fragments de programme sur lesquels appliquer le «checkpointing». Enfin, un objectif à plus long terme du projet, la comparaison de fragments de programmes, repose sur cette transformation.

Nous avons introduit cette transformation dans la plateforme ALI. Techniquement, on a dû résoudre deux problèmes principaux :

- vérifier qu'un fragment donné par l'utilisateur est correct du point de vue du contrôle. Schématiquement : le fragment donné peut-il être isolé en un nouveau sous-programme ?
- déterminer les ensembles minima de variables (arguments ou globales) qui doivent être passés en paramètre, par valeur ou par référence. En d'autres termes, déterminer le plus précisément possible l'*interface* entre l'extérieur et l'intérieur du fragment considéré.

6.5 Interface Web

Mots clés : différentiation automatique, interface, web.

Participants : Laurent Hascoët, Valérie Pascual, Jean-Charles Rihaoui.

L'outil de différentiation a besoin d'une interface. Il peut fonctionner en mode «batch» sans interface, mais même dans ce cas, une interface a posteriori serait très utile. Une interface doit permettre de visualiser le programme différentié, en regard du programme d'origine. Elle doit aussi donner les représentations abstraites que les utilisateurs attendent, comme le graphe d'appel. On comprend l'intérêt de cette interface pour le debug et la validation. Il y a aussi un intérêt pédagogique, pour démythifier la D.A. et convaincre de nouveaux utilisateurs de son intérêt.

Malheureusement, une interface est un élément mouvant et délicat d'une application. Les portages en sont souvent pénibles. Pour contourner ce problème, nous avons exploré la possibilité d'une interface HTML. Les navigateurs web sont omniprésents, et ils assurent eux-même le portage sur des architectures très diverses. Si nous décrivons notre interface sous forme de fichiers HTML, nous bénéficions de ces portages. Le langage HTML lui-même est assez rustique et propre, surtout dans la forme XHTML.

L'autre motivation est celle de l'installation du logiciel. Pour faire une simple démo, il est regrettable de devoir installer un logiciel, avec tous les risques que cela comporte, et surtout le risque que l'installation échoue. Nous voulons donc que l'outil de D.A. soit une servlet installée

dans nos murs, et accessible via le web et l'interface HTML.

A notre connaissance, il n'existe pas sur le web d'outils comparables, par exemple des compilateurs ou des analyseurs de programme accessibles en ligne. Jean-Charles Rihaoui a fait un premier prototype durant son stage. Les resultats sont bons, et Valérie Pascual a repris ce prototype pour développer l'interface web complète.

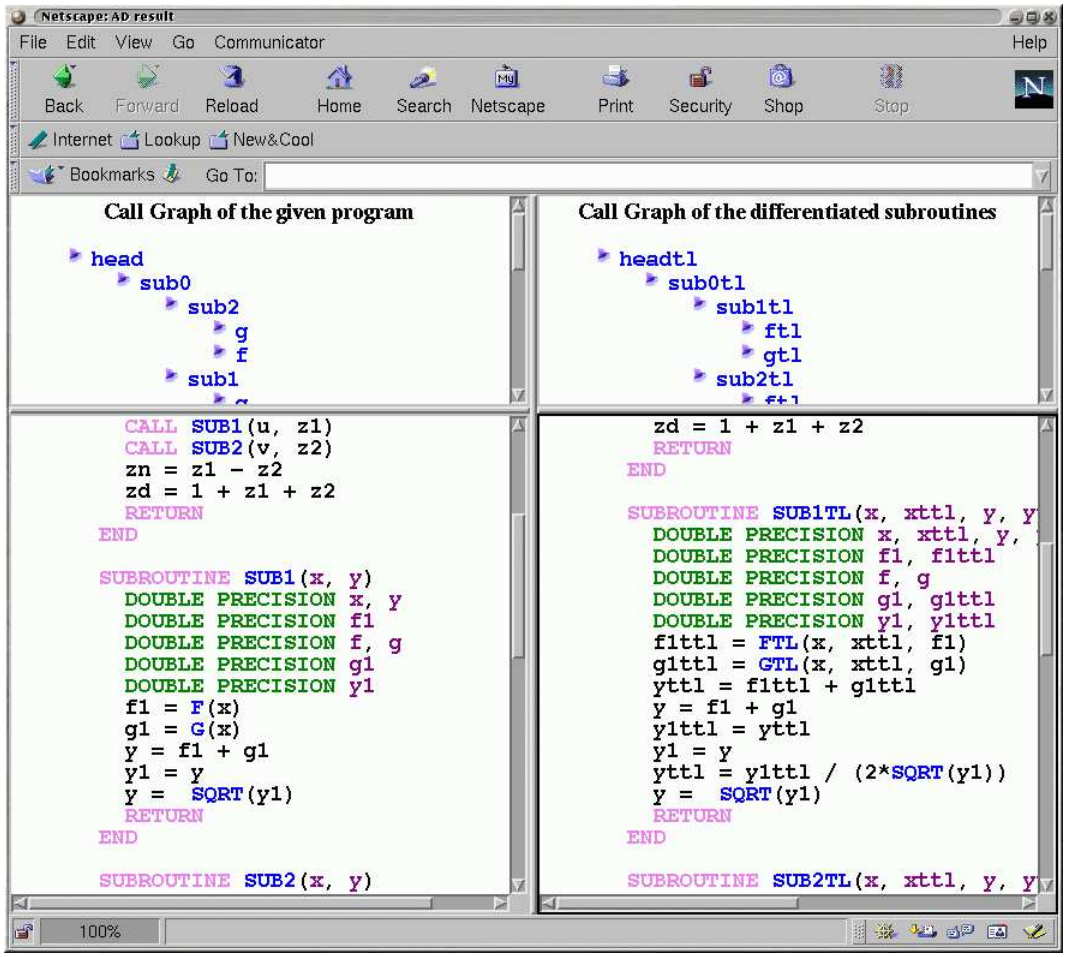

FIG. 1 – Page HTML des résultats de la D.A.

L'interface web comporte une page d'accueil, dans laquelle l'utilisateur est invité à «uploader» ses fichiers source, y compris les include. Il saisit ensuite les noms du sous-programme à différentier, de ses variables dépendantes et indépendantes. Il demande enfin la différentiation, soit en mode direct, soit en mode inverse. Le résultat est présenté dans une nouvelle page, avec un graphisme qui souligne la syntaxe et les liens entre le programme d'origine et le programme différentié. Il est possible alors de «downloader» tout ou partie du programme généré.

Dans la page HTML des résultats de la D.A., présentée sur la figure 1, on utilise des combinateurs `FRAME` pour séparer quatre zones. Le programme initial est à gauche, le programme différentié est à droite. La représentation schématique du graphe d'appel est en haut, le texte

des programmes est en bas. Il s'agit en fait de fichiers HTML. Les attributs graphiques (couleurs,...) sont définis par des style sheets `css`, et mettent en valeur la syntaxe d'une manière dépendante du contexte, c'est à dire plus intelligemment que ne le fait Xemacs par exemple. L'affichage est effectué par un décompilateur FORTRAN développé cette année. L'interface fait correspondre automatiquement chaque instruction d'origine et ses instructions différentiées, au moyen d'ancres HTML. On fait de même correspondre un nœud du graphe d'appel avec le texte du sous-programme correspondant.

6.6 Base d'Exemples et de Tests pour la Démonstration et la Validation

Mots clés : différentiation automatique, différences divisées, mode direct, mode inverse.

Participants : Rose-Marie Greborio, Valérie Pascual.

Rose-Marie Greborio construit une base d'exemples pour la validation de l'outil de D.A. Il y a deux parties principales. D'une part une collection de petits programmes, à utiliser comme tests de non-régression. D'autre part une collection de programmes numériques réels, fournis par nos collègues numériciens de l'INRIA et d'autres centres académiques, sur lesquels on valide la D.A. en comparant avec les différences divisées.

Dans ce cadre, Rose-Marie Greborio développe et teste un mécanisme d'automatisation des tests de non-régression. Elle maintient également une base de données interne des problèmes identifiés et non résolus. Enfin, chaque exemple de programme grandeur nature est accompagné d'une procédure automatique de validation de la différentiation. Cette procédure évalue d'une part les dérivées par différences divisées, d'autre part par différentiation automatique en mode direct, et enfin en mode inverse.

Le but de cette démarche est d'aboutir à un logiciel de qualité, fiable et utilisable à l'extérieur de l'INRIA. Nous devons satisfaire les utilisateurs de l'ancien ODYSSÉE, et aussi de nouveaux utilisateurs et collaborateurs. D'autre part, les exemples d'application sur des programmes réels permettent non seulement de valider l'approche D.A., mais aussi de promouvoir la D.A. auprès d'un public plus habitué aux différences divisées.

A plus long terme, nous voulons diffuser la plate-forme ALI. Pour cela, une première ébauche d'API (Application Programmer Interface) a été définie pour ALI. Elle reste améliorable. Bien entendu, la principale source d'amélioration sera d'écouter les demandes d'utilisateurs exterieurs.

6.7 Optimum Design par gradient

Mots clés : optimum design, optimisation, gradient, différentiation automatique, mode inverse.

Participants : François Courty, Alain Dervieux, Jean-Charles Gilbert [projet Estime], Laurent Hascoët, Bruno Koobus, Bijan Mohammadi [Univ. Montpellier et projet M3N].

L'optimisation en aérodynamique reste un problème difficile et coûteux en temps calcul et le calcul d'un gradient par une méthode potentiellement efficace comme celle de l'état adjoint est un enjeu important en ingénierie. L'obtention de gradients analytiques ou "exacts" est

une tâche très complexe. Notre travail permet de mieux cerner la problématique de la méthode inverse et d'entrevoir une application efficace à l'assemblage de l'état adjoint stationnaire. Nous appliquons nos résultats antérieurs concernant la méthode "iteration-wise adjoining", mise au point par Laurent Hascoët. Nous proposons une tactique de mise en œuvre de cette méthode sous Odyssée, en attendant une implémentation native dans TAPENADE. Cette tactique a été décrite en détail et validée sur un exemple issu de la Mécanique des Fluides dans le cadre du projet Européen Aeroshape. Elle a aussi été expliquée à plusieurs partenaires de Aeroshape notamment lors de leurs visites à Sophia (V. Selmin, Alenia(I), N. Petrapoulou, HCSA(GR), notamment).

Ce travail est présenté dans un article en cours de rédaction avant soumission en revue.

La méthode de l'état adjoint combinée à la différentiation automatique permet d'envisager une approche moderne de l'optimisation de la forme aérodynamique d'un avion. Par exemple on peut appliquer un algorithme de type Programmation Quadratique Séquentielle. Dans ce cadre, toujours pour le projet Aéroshape, nous cherchons à améliorer les algorithmes existants en les adaptant au contexte de calcul très intensif de l'optimisation en Aérodynamique. Dans ce contexte nous travaillons avec Jean-Charles Gilbert sur des approches combinant la résolution "one-shot" et les régions de confiance.

Cette année a aussi commencé une étude sur les adjoints pour les problèmes couplés en coopérations avec Bijan Mohammadi et Charbel Farhat (Université de Boulder Colorado). Une application à l'optimisation d'un avion souple en couplage fluide-structure est en cours de mise en place.

7 Diffusion de résultats

7.1 Relations industrielles et contrats

- Aeroshape est un consortium soutenu par le programme Européen GROWTH dont le but est de faire avancer la capacité des laboratoires de recherche publics et privés dans la réalisation de boucles d'optimisation de formes en aérodynamique. Les partenaires industriels sont les principaux constructeurs d'avions en Europe. La participation de l'INRIA est de type méthodologique, et consiste à proposer d'une part un outils et une méthode pour l'assemblage d'adjoints analytiques et d'autre part des algorithmes d'optimisation adaptés au contexte doublement spécifique de l'assemblage par Différentiation Automatique et des résolutions d'états et adjoints très coûteuses en Aérodynamique complexe (Navier-Stokes, turbulent, 3D). Voir le site :
  `http://www-sop.inria.fr/tropics/aeroshape.html`.
- Le programme "Avion Supersonique" du Ministère de la Recherche soutient la proposition "Optimisation d'un avion souple". Dans ce projet en cours de démarrage, les avancées réalisées dans la mise en place de boucle d'optimisation avec état adjoint (notamment dans Aeroshape) seront étendues à un contexte d'optimisation multi-disciplinaire (fluide, structure, bang sonique) et multi-point (plusieurs régimes de croisière : subsonique, supersonique). Ce programme nous associe avec l'université de Montpellier (Prof. B. Mohammadi) et Dassault Aviation qui proposera des géométries d'avion d'affaire supersonique.

7.2 Participation à des colloques, séminaires, invitations

- Réunion du projet Aeroshape à Sophia-Antipolis. Présentation par L. Hascoët d'une optimisation du mode inverse de Différentiation Automatique pour les boucles d'assemblage en calcul scientifique.
- Invitation de L. Hascoët à venir présenter les travaux du projet devant l'équipe de Christian Bischof à l'université de Aachen, les 29 et 30 novembre. Démonstration de l'outil TAPENADE.
- Invitations de A. Dervieux à ECCOMAS'2001 (Swansea), à Amflow2001 (Heidelberg), à un workshop franco-australien (Melbourne), visite avec séminaire à l'Institut Liapunov et à l'IMM (Moscou).

7.3 Animation de la communauté scientifique

Laurent Hascoët est co-éditeur du livre [8], qui présente l'état de l'art en Différentiation Automatique, à travers les analyses des spécialistes du domaine et une sélection d'articles présentés à la conférence AD2000. Cette conférence a eu lieu à Nice du 19 au 23 juin 2000.

8 Bibliographie

Ouvrages et articles de référence de l'équipe

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

[2] P. DUTTO, C. FAURE, S. FIDANOVA, « Automatic differentiation and parallelism », *in : Proceedings of Enumath 99, Finland*, 1999.

[3] C. DUVAL, P. ERHARD, C. FAURE, J. GILBERT, « Application of the Automatic Differentiation Tool `Odyssée` to a system of thermohydraulic equations. », *Numerical Methods in Engineering*, 1996, p. 795–802, Proceedings of ECCOMAS'96.

[4] C. FAURE, Y. PAPEGAY, « Odyssée User's Guide Version 1.7 », *Rapport technique n°224*, INRIA, 1998, `http://www.inria.fr/rrrt/rt-0224.html`.

[5] L. HASCOËT, « Parallelization of Finite Element Codes with Automatic Placement of Communications », *rapport de recherche n°3646*, Inria, `http://www.inria.fr/RRRT/RR-3646.html`.

[6] L. HASCOËT, « Automatic Placement of Communications in Mesh-Partitioning Parallelization », *ACM SIGPLAN Notices 32*, 7, 1997, p. 136–144, Proceedings of 6^{th} ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.

[7] M. TADJOUDDINE, C. FAURE, F. EYSSETTE, « Sparse Jacobian Computation in Automatic Differentiation by Static Program Analysis », *in : Static Analysis*, G. Levi (éditeur), *Lecture Notes in Computer Science, 1503*, Springer-Verlag, p. 311–326, septembre 1998.

Livres et monographies

[8] G. CORLISS, C. FAURE, A. GRIEWANK, L. HASCOËT, U. NAUMANN, *Automatic Differentiation of Algorithms, from Simulation to Optimization*, *LNCSE*, Springer, 2001.

Articles et chapitres de livre

[9] L. Hascoët, « A method for automatic placement of communications in SPMD parallelisation », *Parallel Computing Journal*, 27, 2001, p. 1655–1664.

[10] C. Held, A. Dervieux, « One-shot airfoil optimisation without adjoint », *Computers and Fluids, sous presse, 2001.*

Rapports de recherche et publications internes

[11] F. Courty, A. Dervieux, B. Koobus, L. Hascoet, « Reverse automated differentiation for optimum design : from adjoint state assembly to gradient computation », *rapport de recherche n°4363*, INRIA, à paraître, `http://www.inria.fr/rrrt/rr-4363.html`.

[12] L. Hascoët, « The Data-Dependence Graph of Adjoint Programs », *rapport de recherche n°4167*, INRIA, 2001, `http://www.inria.fr/rrrt/rr-4167.html`.