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]
Rose-Marie Greborio [Ingénieur spécialiste industriel]

Post-doc
Mariano Vazquez [du 1/1/2002 au 30/11/2002]

Chercheurs doctorants
François Courty [Boursier INRIA]
Guillaume Thibaudeau [Boursier INRIA]

Visiteurs
Tatyana Kozubskaya [Institute of Mathematical Modeling, Moscou, Russie, du 1/2/2002 au 30/4/2002]
Ilya Abalakin [Institute of Mathematical Modeling, Moscou, Russie, du 1/2/2002 au 28/2/2002]

Collaborateur extérieur
Bruno Koobus [Conseiller scientifique, université de Montpellier]

Stagiaires
Mauricio Araya-Polo [DEA université de Nice, depuis le 1/11/2002]
Jeremy Buisson [Elève INSA Rennes, du 17/6/2002 au 30/8/2002]

2. Présentation et objectifs généraux

Le projet TROPICS, multidisciplinaire, comporte deux volets :

- D'une part, nous étudions les techniques logicielles d'analyse et de transformation semi-automatiques de programmes. Cela nous a conduit par le passé à développer des algorithmes de parallélisation semi-automatique. A l'heure actuelle, la transformation qui mobilise nos efforts est la Différentiation Automatique (DA). La DA transforme un programme pour qu'il calcule certaines dérivées de la fonction calculée par le programme initial. En particulier, la DA permet de calculer des gradients, grâce au mode dit « inverse ». Ce mode inverse demeure toutefois très délicat à mettre en œuvre.
- D'autre part, nous étudions les méthodes de Calcul Scientifique, principalement en mécanique des fluides numérique, et les techniques d'optimisation. Ces études sont appliquées à des problèmes réels soumis par nos partenaires industriels. Les techniques d'optimisation font appel à des notions telles que dérivées ou gradients. Les calculs sont lourds et coûteux, et ont donc un grand besoin des techniques d'amélioration des performances, telles que la parallélisation.

La deuxième discipline du projet (optimisation en calcul scientifique) est donc tout à la fois la motivation et le domaine d'application de la première discipline du projet (transformation de programmes et gradients par DA). Les travaux en calcul scientifique touchent notamment :

1. la simulation Numérique : on se focalisera sur la Simulation Numérique en Mécanique des Fluides Compressibles, et sur les couplages multidiscipline,

2. le Contrôle Optimal avec une focalisation sur la Conception Optimale de Formes Aérodynamiques, y compris dans un cadre multidisciplinaire,
3. les algorithmes d'Optimisation avec une focalisation sur l'adaptation des algorithmes de Programmation Quadratique Séquentielle au traitement des gros problèmes issus de (2),
4. les préconditionneurs fonctionnels aptes à améliorer l'efficacité des algorithmes utilisés dans (2) et (3),
5. la manipulation des modèles réduits pour l'optimisation mono- et multi-disciplinaire,
6. l'optimisation de maillage.

Les points (2)(3)(4)(6) touchent aux méthodes d'adjoints en Contrôle Optimal et donc à la différentiation « inverse », le point (5) à la différentiation « directe ».

Les travaux en DA touchent notamment :

- les optimisations mémoire pour le mode inverse (calcul de gradients) : On se focalise sur la détection des sauvegardes inutiles de valeurs intermédiaires, ainsi que sur les stratégies stockages/recalcul indispensables pour calculer le gradient de gros programmes,
- l'application des outils conceptuels de compilation, comme l'analyse du flot des données, pour optimiser les programmes différentiés en réordonnant les instructions, en particulier pour le mode multi-directionel,
- les techniques de validation des dérivées obtenues par DA,
- 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.

Ces recherches se traduisent dans un outil, TAPENADE, développé et maintenu par le projet. Les fonctionnalités de TAPENADE sont dorénavant strictement supérieures à celles de l'ancien outil de DA ODYSSÉE, qui n'est donc plus maintenu. TAPENADE est un outil de DA, mais aussi une plate-forme 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.

TAPENADE est distribué librement sur notre site ftp://ftp-sop.inria.fr/tropics. Il est aussi accessible directement sous forme d'un serveur web à l'adresse http://tapenade.inria.fr:8080/tapenade/index.jsp. A ce jour, nous avons enregistré de nombreuses connexions au serveur ou téléchargements de TAPENADE, parfois pour évaluation, souvent pour effectuer un travail réel. En particulier, nous avons des utilisateurs réguliers aux universités de Dresde et Aix-la-Chapelle (Allemagne), au RMCS Cranfield et à l'université de Reading (Grande Bretagne), l'université Duke et l'ANL Argonne (Etats Unis), au CEA Cadarache et à l'INRIA Rhône-Alpes/IMAG à Grenoble. Toute la documentation disponible sur TAPENADE, incluant une présentation de la DA et un tutorial, sont accessibles en ligne sur le site du projet http://www-sop.inria.fr/tropics/.

Nous maintenons des relations avec les universités de Dresde (Allemagne), Aix-la-Chapelle (Allemagne) et Hatfield (GB). Nous collaborons, dans le cadre de projets européens, avec Dassault-Aviation, BAE (Grande Bretagne), Alenia (Italie). En particulier, ces industriels utilisent notre outil TAPENADE.

3. Fondements scientifiques

3.1. Différentiation Automatique

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

Participants : Mauricio Araya-Polo, Jeremy Buisson, François Courty, Rose-Marie Greborio, Laurent Hascoët, Valérie Pascual, Mariano Vazquez.

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 DA, 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 (DA), 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 DA a une longue tradition à l'INRIA [35].

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

- Par surcharge des opérateurs arithmétiques (outils ADOL-C [37], ADOGEN [47]). Cette approche permet des développements rapides,
- Par transformation de source à source (outils ODYSSÉE [4], ADIFOR [26], TAMC [34], PADRE2 [41]) ; cette approche demande le développement d'outils complexes, mais permet une plus grande flexibilité.

On trouvera une liste complète des systèmes actuels de DA à l'adresse : http://www.autodiff.org, qui est le site de la communauté des projets travaillant sur la DA. 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 [39]. Divers raffinements intéressants sont étudiés, 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 dans 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

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

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 [39].
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 [29]. Les compromis stockage/recalcul font aussi l'objet de recherches actives [36], [38], [28].

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.

3.2. Parallélisation

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

Participants : Laurent Hascoët, 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 le calcul des dépendences de données.

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. Pour être efficace, la parallélisation doit s'appuyer sur une analyse fine des *dépendances de données*, 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, par exemple le modèle SPMD.

L'aide à la parallélisation SPMD est davantage guidée par les demandes des utilisateurs numériciens. 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 [9] [7], pour le placement optimal des appels aux routines de communication.

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*, *analyse de flot de données*, *graphe de dépendances*.

Participants : Mauricio Araya-Polo, Jeremy Buisson, Rose-Marie Greborio, Laurent Hascoët, Valérie Pascual.

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.

analyse du flot des données Méthode d'analyse des programmes informatiques qui, au lieu de s'appuyer directement sur le texte du programme, étudie la manière dont les *valeurs* des variables se propagent au cours du programme. Cette propagation est souvent représentée par le *graphe de dépendances* des données, et c'est sur ce graphe que s'appuient les analyses du flot de données.

graphe de dépendances des données 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 [22]. 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 Cornell Synthesizer [46] et CENTAUR [27][50]. 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 [40][25], 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 [30], 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*, qui représente le flot des données. Suivant l'utilisation (parallélisation, différentiation, comparaison de programmes...), on en utilise diverses variantes, dont le *data-flow graph*, le *data-dependence graph* [42] [23], le *program dependence graph* [33] ou le *dependence flow graph* [44]. L'utilisation de ces graphes dans le cas particulier de la parallélisation est décrit dans [51], [32], [23] ou [31] et, pour la comparaison de sous-programmes, dans [45][24]. 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 : Ilya Abalakin, François Courty, Alain Dervieux, Laurent Hascoët, Bruno Koobus, Tatyana Kozubskaya, Mariano Vazquez.

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 permet la mise en œuvre par une partie de l'équipe d'applications démonstratives de haute complexité, qui concernent *l'optimisation de forme en aérodynamique*.

Les outils de Différentiation Automatique développés et maintenus par Tropics sont appliqués dans ce contexte de facon (1) à génerer automatiquement des parties de code calculant des dérivées plutôt que de les écrire, (2) à fournir aux développeurs de TAPENADE des informations sur le fonctionnement des strategies de DA existantes, et (3) à contribuer à la spécification de nouvelles stratégies.

Les problèmes 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 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'application

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 linearisation au premier ordre de systèmes complexes,
- l'adaptation de maillages,
- 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 [43] [49],
- les méthodes d'optimisation de formes, les méthodes d'optimisation sous contraintes, et plus généralement tous les algorithmes basés sur une linéarisation locale,
- les modèles réduits construits à partir de la différentiation du modèle numérique complet.

Parmi ces applications, nous allons détailler celles que nous avons étudiées de plus près.

4.2. Optimisation multidisciplinaire

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 TAPENADE est une approche féconde.

4.3. Problèmes inverses

Les problèmes inverses consistent a retrouver les valeurs de paramètres cachés, non directement mesurables, à partir de valeurs mesurables qui dépendent de ces valeurs cachées. Par exemple, il peut s'agir de retrouver la forme du fond de l'océan à partir de mesures sonar, ou bien à partir de mesures des courants marins en surface. Il peut s'agir aussi d'ajuster la valeur de paramètres clés d'un modèle, pour que ce modèle s'accorde le mieux possible avec les mesures. Considérons par exemple 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 ? Il s'agit d'un problème similaire à l'estimation des paramètres, communément appelé l'« assimilation des données ». La réponse à ces problèmes inverses 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 [48].

4.4. Linearisation au premier ordre

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, par exemple en aéroacoustique, 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.

4.5. Adaptation de maillage

Depuis quelques années on s'est aperçu que certaines erreurs d'approximation s'expriment à l'aide d'un état adjoint. L'écriture de l'équation adjointe est une tâche fastidieuse et propice aux erreurs de programmation. La Différentiation en mode inverse constitue donc une bonne alternative. Le maillage adaptatif résulte alors d'un processus d'optimisation. Ces démarches sont de plus en plus intégrées dans des boucles de conception optimale de telle sorte que l'optimisation porte à la fois sur le contrôle et la qualité d'approximation. Le but à terme est la mise au point d'algorithmes livrant un Contrôle Optimal à précision prescrite à l'avance. Une autre application est l'accroissement par un terme correcteur de la précision d'une caractéristique de la solution calculée.

4.6. Modèles réduits

La Différentation Automatique est couramment utilisée pour la mise au point de modèles réduits. Les modèles réduits sont des modèles numériques à faible nombre d'inconnues et très peu coûteux en temps calcul. Ils sont utilisés soit dans des contextes multidisciplinaires, soit dans des boucles d'optimisation intensive (algorithmes génétiques par exemple). Les modèles réduits sont généralement moins précis que les modèles traditionnels, typiquement en éléments finis. Ils en sont souvent déduits par des approximations de type linéarisation ou série de Taylor. C'est dans la préparation de ces approximations qu'intervient la Différentiation Automatique.

5. Logiciels

5.1. ALI

Participants : Laurent Hascoët [correspondant], Mauricio Araya-Polo, Jeremy Buisson, Rose-Marie Greborio, Valérie Pascual.

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, l'outil de Différentiation Automatique du projet, TAPENADE, est bâti au-dessus d'ALI.

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 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 amélioré la fiabilité, et accepté de nouveaux éléments syntaxiques de FORTRAN90 (Types structurés, Structure `switch/case`).

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 ébauche d'API (Application Programmer Interface) existe, 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], Mauricio Araya-Polo, Jeremy Buisson, Rose-Marie Greborio, Valérie Pascual.

TAPENADE est l'outil de DA développé par le projet. TAPENADE est construit en Java au-dessus de la plateforme ALI.

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.

TAPENADE met en œuvre un certain nombre de techniques et d'analyses pour produire un résultat efficace. Outre un calcul de dépendances des données spécifique de la DA, et la détection des variables actives, déjà validés dans ODYSSÉE mais grandement améliorés 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 6.2.

- 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.
- L'analyse du flot des données, localement à chaque bloc d'instructions, pour regrouper les instructions dérivées et par conséquent améliorer les performances du mode tangent multi-directionel. Cela est décrit dans la section 6.4.
- L'analyse In-Out, classique en compilation, 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. La dernière version 2.01 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. Il est aussi possible de télécharger TAPENADE pour une installation locale. L'article [15] présente TAPENADE, en mettant l'accent sur le mode inverse et son utilisation pour résoudre les problèmes inverses.

6. Résultats nouveaux

6.1. Optimisation des programmes différentiés par fusion d'instructions et propagation des dérivées remarquables

Mots clés : *Analyse de flot de données*, *graphe de dépendance des données*, *compilation*.

Participants : Laurent Hascoët, Valérie Pascual.

Nous avons étudié plusieurs manières de produire des programmes différentiés plus concis et plus efficaces. Plus précisément, nous avons envisagé :

- de regrouper les instructions, et plus particulièrement les instructions dérivées en mode inverse, qui sont souvent des séquences de réinitialisations à zéro et d'incrémentation.
- de propager à l'intérieur du programme l'information selon laquelle une variable a une dérivée remarquable, soit nulle, soit inutilisée.

En ce qui concerne les instructions du mode inverse de la différentiation, nous avons besoin de repérer les instructions dérivées qui effectuent des opérations que l'on peut regrouper. Rappelons que les instructions dérivées en mode inverse pour une affectation classique, telle que

```
y = 2*a*b + 5
```

forment la séquence d'instructions suivante :

```
ā = ā + 2*b*ȳ
b̄ = b̄ + 2*a*ȳ
ȳ = 0.0
```

où ȳ, par exemple, est la variable dérivée de y en mode inverse. On remarque alors que, si l'on regroupe les incrémentations d'une variable telle que ā, et une réinitialistion précédente de ā à zéro, alors on améliore le programme dérivé. Bien qu'un compilateur sache d'ordinaire effectuer de tels regroupements lorsque les instructions sont proches, il est bien plus puissant de le faire au cours de la différentiation, c'est à dire dès la création de ces instructions d'incrémentation et d'initialisation. En utilisant une analyse de flot de données, on peut trouver le meilleur regroupement possible des instructions dérivées.

En ce qui concerne les variables à dérivée remarquable, nous disposons maintenant dans TAPENADE de l'analyse des variables *actives* dans les deux directions. Plus précisément, TAPENADE détecte les variables *utiles*, dont la valeur influence (de manière différentiable) celle des variables de sortie, et les variables *variées*,

dont la valeur dépend (de manière différentiable) de celle des variables d'entrée. Une occurrence d'une variable n'est active que si elle est à la fois *utile* et *variée*. Que l'on soit en mode tangent ou inverse, seules les instances actives de variables ont besoin d'une variable dérivée :

- En mode tangent, si v n'est pas variée, alors sa dérivée $\dot{v} = 0$, et on peut remplacer toutes les occurences de $\dot{v}$ par zéro. Si v n'est pas utile, alors nous savons qu'il n'y a pas d'occurence de $\dot{v}$ dans la suite, et il est inutile de calculer $\dot{v}$.
- En mode inverse, si v n'est pas variée, alors l'éventuelle valeur de $\bar{v}$ n'influe pas sur le gradient final par rapport aux variables d'entrée. Il est donc inutile de calculer $\bar{v}$. Si v n'est pas utile, alors il est certain que $\bar{v}$, gradient partiel du résultat final par rapport à v, est nul. On peut remplacer toutes les occurences de $\bar{v}$ par zéro.

Il reste à définir comment ces règles simples s'étendent à un graphe de flot de contrôle, pour un programme arbitraire. C'est ce que nous avons fait cette année. Nous avons implémenté cela dans la version 2.0 de TAPENADE. Le résultat est une réduction très importante des instructions de réinitialisation de variables dérivées, car nombre d'entre elles sont effectivement inutiles.

6.2. Formalisation des analyses d'activité et de sauvegarde des valeurs intermédiaires

Mots clés : *Analyse statique*, *Graphe de flot*, *compilation*.

Participants : Laurent Hascoët, Uwe Naumann [Argonne National Laboratory, Argonne, Illinois].

Dans Odyssée 1.7, puis dans TAPENADE, nous avions implémenté l'analyse d'activité, ainsi qu'une analyse spécifique du mode inverse, nommée TBR. Ces deux analyses statiques étaient spécifiées informellement. Uwe Naumann avait par ailleurs prouvé certaines propriétés de l'analyse TBR sur des programmes bien structurés : en particulier qu'un point fixe sur le flot de contrôle n'était pas nécessaire car ce point fixe est toujours atteint en une seule itération.

Cette année, nous avons remis à plat la spécification de ces deux analyses, pour des flots de contrôle arbitraires à la fois dans le cas structuré et non structuré. Nous avons utilisé pour cela le formalisme classique des équations de data-flow. Cela nous a donné une description à la fois plus formelle et plus claire. En outre, ce formalisme permet une nouvelle preuve, beaucoup plus concise, de la propriété ci-dessus de l'analyse TBR.

Cette formalisation nouvelle a donné lieu à un article commun [16], pour un numero special du journal « Future Generation Computer Systems », à la suite de la conférence ICCS'02, à Amsterdam.

6.3. Etude de l'influence des pointeurs en Différentiation

Mots clés : *Analyse de pointeurs*, *Analyse statique*, *compilation*.

Participants : Jeremy Buisson, Laurent Hascoët.

Dans le cadre du stage d'été de Jeremy Buisson, élève de 2ème année à l'INSA de Rennes, nous avons étudié la Différentiation Automatique des programmes qui utilisent des pointeurs. Avant de spécifier la DA de programmes avec pointeurs, nous avons commencé par construire un ensemble de petits programmes avec pointeurs, qui illustrent les utilisations les plus classiques.

Nous avons tout d'abord séparé deux problèmes : l'allocation de la mémoire et l'utilisation des pointeurs. Les deux questions sont liées, mais nous avons préféré étudier les sous-programmes qui ne font aucune allocation. En d'autres termes, nous supposons que les allocations et de-allocations mémoire ont été repoussées *en dehors* des sous-programmes à différentier. Cette restriction devra être levée à moyen terme.

Ensuite, sur les exemples construits, nous avons défini à quoi devrait ressembler le résultat de la différentiation, en mode tangent et en mode inverse. Un résultat intéressant est qu'on doit introduire une notion nouvelle de « pointeur dérivé ». Plus exactement, lorsque le pointeur `p` peut pointer vers les variables `a` ou `b`, on doit introduire sa dérivée `p'`, qui peut pointer vers `a'` ou `b'`, dérivées respectives de `a` et `b`.

D'autre part, on s'aperçoit qu'il est très vite nécessaire d'avoir à sa disposition une analyse statique des destinations possibles des pointeurs. En effet, si l'on ne sait rien de ces destinations, on devra supposer qu'un pointeur peut adresser toute variable, et cela amène, particulièrement en mode inverse, à sauvegarder une trop grande quantité de valeurs.

Côté développement, un prototype a été construit, incluant l'analyse syntaxique des programmes avec pointeurs, et réciproquement leur regénération. Le prototype implémente une version très simplifiée de l'analyse des destinations de pointeurs, et surtout une différentiation en mode direct qui tient compte des pointeurs. Ce prototype, intégré dans une version de TAPENADE donne satisfaction sur de petits exemples, mais est tout de même encore très limité.

6.4. Différentiation multi-directionelle avec fusion de boucles

Mots clés : *Différentation en mode tangent*, *Matrice Jacobienne*, *fusion de boucles*.

Participants : Laurent Hascoët.

L'outil de DA ADIFOR propose un mode de différentiation particulier, nommé *vectoriel*. Dans ce mode, on demande non pas la différentiation suivant une direction particulière dans l'espace des données, mais suivant *plusieurs* directions simultanément. Cela est utile par exemple dans les méthodes de gradient conjugué. Cela permet aussi de construire progressivement toute la matrice Jacobienne.

Il est bien entendu possible d'obtenir ces n dérivées en exécutant n fois le programme différentié en mode *tangent*. Mais il est probablement plus efficace de calculer les n dérivées directionnelles pendant la même exécution du programme initial. La raison principale est qu'on ne répète pas les opérations du programme initial, qui ne dépendent pas de la direction de différentiation particulière choisie. Dans le mode vectoriel, chaque instruction dérivée devient une boucle. Chaque itération de cette boucle correspond à une direction de différentiation.

Nous avons développé un tel mode à l'intérieur de TAPENADE. Par rapport à ADIFOR, nous avons introduit une amélioration permise par l'architecture interne de TAPENADE. Par une analyse de flot des données, nous construisons un graphe de dépendance dont les noeuds sont les instructions, initiales ou différentiées. Grâce à ce graphe, nous regroupons les instructions différentiées quand c'est possible. Nous pouvons alors fusionner les boucles sur les n directions de différentiation. Le programme obtenu est alors plus efficace car le coût constant dû aux boucles est réduit. Sur nos premiers exemples, on observe une réduction importante du nombre de boucles. On peut gagner un facteur supérieur à 10.

L'équipe de Chris Bischof à Aix-la-Chapelle propose de paralléliser ces boucles (en utilisant OpenMP). Or l'overhead de lancement des boucles parallèles OpenMP est loin d'être négligeable. Nous pensons donc que le mode multi-directionel de TAPENADE peut améliorer notablement les performances de la parallélisation.

6.5. Routines externes et extensions Fortran90

Mots clés : *Fortran90*, *interface utilisateur*.

Participants : Rose-Marie Greborio, Laurent Hascoët.

Nous présentons ici les extensions apportées cette année aux modes standards de TAPENADE (tangent et inverse).

En Différentiation Automatique, les routines externes, ou plus généralement les routines dont le source n'est pas donné, posent un problème. Comme les analyses préliminaires à la DA sont des analyses globales, elles ont besoin des résultats des analyses de tous les sous-programmes appelés par le sous-programme à différentier principal.

Si l'on ne connaît pas le source d'un sous programme, on est amené à faire des hypothèses conservatives sur ce sous-programme. Cela amène une baisse des performances du résultat final. Par exemple, comme on est obligé de considérer que tout paramètre en sortie peut dépendre au sens de la différentiation de tout paramètre en entrée, on doit :

- prévoir un paramètre dérivé pour chaque paramètre d'origine,

- supposer que tous les paramètres de sortie sont actifs (dérivée non nulle) si au moins l'un des paramètres d'entrée est actif,
- supposer que chaque paramètre peut être modifié durant l'exécution du sous-programme,
- en mode inverse, sauvegarder tous les paramètres du sous-programme si l'on compte l'exécuter à nouveau (tactique nommée *checkpointing*).

Il faut donc éviter d'avoir à faire ces hypothèses conservatives.

Nous avons défini un formalisme permettant à l'utilisateur, sans donner le source du sous-programme, de déclarer pour les paramètres de ce sous-programme, leur type, le fait qu'ils soient lus ou écrits durant le sous-programme, et les dépendances reliant la valeur en sortie aux valeurs en entrée. En gros, nous permettons à l'utilisateur de fournir manuellement les attributs qui seraient synthétisés par l'outil si le source était donné. La différentiation utilise ces informations pour produire un code plus efficace.

Parallèlement, les modes de base de TAPENADE ont été étendus pour traiter des constructions présentes en Fortran95 et non en Fortran77. TAPENADE accepte et différentie désormais les instructions `switch-case`, les variables de type `record` (« derived type » dans le jargon Fortran95), les variables de type `COMPLEX` ainsi que les notations vectorielles.

6.6. Interface et messages d'erreur par liens HTML

Mots clés : *interface utilisateur*, *messages d'erreur*, *HTML*.

Participants : Valérie Pascual.

En préliminaire à la différentiation proprement dite, TAPENADE effectue des analyses statiques sur les programmes. Ces analyses produisent souvent des messages, soit d'erreur, soit d'avertissement. Ces messages sont particulièrement importants en DA. En effet, alors que ces erreurs sont souvent anodines et bien rattrappées par le compilateur, la différentiation des ces programmes incorrects produit des programmes « vraiment très incorrects », et même souvent faux ! Un exemple typique est l'aliasing, décrit dans notre FAQ.

L'utilisateur final de TAPENADE doit donc vérifier soigneusement les messages produits par l'outil, et nous avons étendu l'interface pour l'y aider. L'interface de TAPENADE est construite en HTML. Nous avons ajouté à cette interface un cadre supplémentaire qui affiche les messages d'erreur. Des liens hypertexte (des *« ancres »* HTML) implémentent la correspondance entre les messages d'erreur et la ligne en cause dans le source, et ce dans les deux directions. La figure 1 montre cette extension de l'interface.

6.7. Optimum Design par gradient

Mots clés : *Optimum design*, *Gradient*, *Modèle adjoint*.

Participants : François Courty, Bruno Koobus, Alain Dervieux, Laurent Hascoet, Mariano Vázquez.

L'optimisation est la suite naturelle de la simulation. L'optimisation en aérodynamique reste un problème difficile et coûteux en temps calcul. Le mot-clé de l'approche choisie est l'« état adjoint ». Il utilise parfaitement les avancées en Différentiation Automatique -notamment en mode inverse- réalisées par le projet.

L'obtention de gradients analytiques pour des codes de simulation numérique est une tâche très complexe ; le travail de l'équipe Tropics a permis de mieux cerner la problématique de la méthode inverse et d'entrevoir une application efficace à l'assemblage de l'état adjoint « stationnaire » via une approche utilisant le « checkpointing ». La méthode « iteration-wise adjoining » [6], a été construite en détail et validée sur un exemple issu de la Mécanique des Fluides dans le cadre du projet Européen AEROSHAPE.

Ce travail, dont une première version est publiée dans [18], est présenté dans un article en cours de soumission en revue.

Par ailleurs, la poursuite de ces mêmes travaux sur des codes 3D dans le cadre du projet du Comité d'Orientation Supersonique a permis de tester TAPENADE.

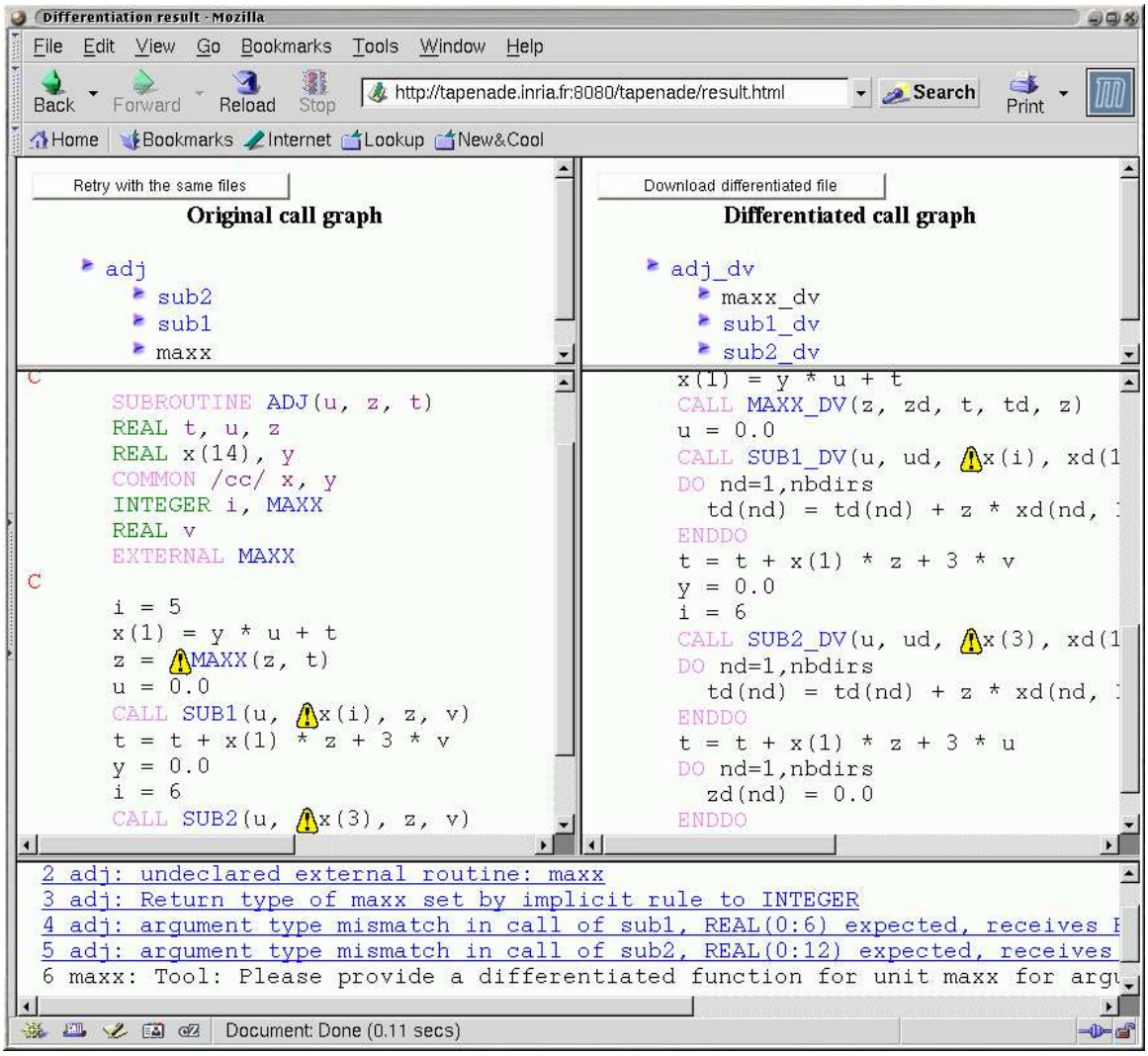

Figure 1. Interface de TAPENADE pour les messages

L'efficacité de l'approche est conditionnée par la combinaison avec un ensemble d'algorithmes qu'il faut souvent profondément transformer pour obtenir un démonstrateur efficace comme le démonstrateur ALYA développé pour le projet COS-Supersonique, et permettant l'optimisation de la forme d'un avion supersonique, voir [21]. Les autres ingrédients de ALYA sont analysés dans les paragraphes suivants.

6.8. Fluide-structure et Optimisation couplée

Mots clés : *optimisation*, *mutidiscipline*, *fluide*, *structure*.

Participants : Bruno Koobus, Alain Dervieux, Mariano Vázquez, Eric Schall, Bijan Mohammadi [université de Montpellier], Charbel Farhat [université de Boulder, Colorado].

L'optimisation de la forme d'un avion supersonique (projet du Comité d'Orientation Supersonique du Ministère de la Recherche) nécessite en pratique de prendre en compte la déformation causée par l'écoulement d'air autour de l'avion.

Nous avons proposé une nouvelle méthode d'optimisation avec couplage faible mais convergeant vers la solution optimale en couplage fort. Cet algorithme permet d'économiser sur les calculs trop importants qui résulteraient d'une approche en couplage fort et représente sur les problèmes considérés une option particulièrement efficace.

Un article sur cette méthode est en cours de rédaction.

Nos travaux en fluide-structure sont motivés par nos activités en optimisation dans le cadre de Tropics, par d'anciennes activités dans Sinus, et par nos activités en Mécanique des Fluides situées dans SMASH.

Plusieurs conférences et un rapport ont présenté des travaux récents dans ce domaine ([14], [13], [19]). Nous avons aussi clarifié les problèmes de conservation d'energie dans le cadre d'écoulements compressibles en maillage mobile (un rapport INRIA en préparation).

Enfin, suite à la visite du sous-préfet Michel Oster des Pyrénées Atlantiques, une importante coopération sur l'étude du « Dirigeable du futur », via le projet FIRST va nous associer avec Adour Space Technology et l'université de Pau (Eric Schall).

6.9. Optimisation par One-Shot et régions de confiance

Mots clés : *optimisation*, *gradient*, *Modèle adjoint*, *Programmation quadratique séquentielle*, *Régions de confiance*, *Quasi-Newton*.

Participants : François Courty, Alain Dervieux, Jean-Charles Gilbert [Projet Estime].

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 sur des approches combinant la résolution « one-shot » et les régions de confiance.

Cette nouvelle méthode a été présentée dans les réunions contractuelles AEROSHAPE ainsi que dans la conférence JP60 [12] et fait l'objet d'un article en cours de soumission [11].

6.10. Optimisation multiniveau

Mots clés : *optimisation*, *multiniveau*.

Participants : François Courty, Bruno Koobus, Alain Dervieux, Mariano Vázquez.

L'étude 3D actuelle fait suite à divers travaux dont certains en 2D ont été publiés cette année [17]. L'optimisation de forme industrielle utilisait jusqu'à récemment un petit nombre de paramètres (typiquement 10 à 20) définissant la forme aérodynamique. Dans l'option « CAD-free », la famille de formes considérée est obtenue par déformation arbitraire de la forme initiale. Le nombre de paramètres en 3D passe à un intervalle de 10.000 à 100.000. Les méthodes quasi-Newton non-préconditionnées sont impuissantes à

résoudre efficacement des problème de cette taille. Nous proposons une nouvelle famille de préconditionneurs de type multiniveau additif applicable à des maillages arbitraires. La technologie de base est la méthode d'agglomération.

Les fondements de la méthode ont été présentés dans la conférence de F. Courty à Heidelberg et le seront en détail dans un article en préparation. Son application dans le cadre de l'optimisation d'un avion supersonique (projet du Comité d'Orientation Supersonique du Ministère de la Recherche) est décrite dans un rapport [21] et dans un article soumis.

6.11. Optimisation et Modèles réduits

Mots clés : *Modèle réduit*, *Décomposion en modes orthogonaux*, *optimisation*.

Participants : Alain Dervieux, Guillaume Thibaudeau, Mariano Vázquez.

Les modèles aux éléments finis (ou volumes finis) sont de plus en plus précis mais toujours assez longs en temps calcul : typiquement une ou plusieurs heures sur 16 processeurs d'un calculateur parallèle de taille moyenne. L'optimisation par gradient ajoute typiquement un ordre de grandeur à ce chiffre et les démarches d'optimisation évolutionnaire deux ordres de grandeur. Dans les deux cas, nous sommes loin d'une réponse « en temps réel » comme le souhaitent les « décideurs ». Pour ce besoin de réponse quasi-instantanée, il s'agira de metre au point des modèles réduits utilisant une base de données de calculs déja réalisés.

Dans le cadre d'un contrat avec Dassault-Aviation, nous avons étudié des modèles aptes à la mise en place de boucle de conception optimale dans le cas stationnaire. nous avons mis au point une approche exploitant notre savoir-faire en discrétisation multi-niveau par agglomération, ainsi que notre expérience en différentiation de solutions d'équations aux dérivée partielle. En pratique, on utilisera la Différentiation Automatique, de préférence aux différences divisées plus coûteuses et moins précises.

Ces techniques nous ont permis de bâtir un prototype de bibliothèque substituable aux noyaux fluides et rendant une réponse 100 à 1000 fois plus rapide. Les déformations de la géométrie sont paramétrées sur une base hiérarchique du type de celle utilisée pour la mise en place du préconditionneur multiniveau. Les perturbations résultant de l'écoulement 3D sont déduites par une dérivation et stockées une fois pour toutes. A chaque appel du calcul d'écoulement simplifié, la perturbation géométrique à analyser est projetée sur la base géométrique et on en déduit l'écoulement par linéarité.

Un rapport de contrat a été livré sur ce thème.

6.12. Optimisation et maillage adaptatif

Mots clés : *Maillage adaptatif*, *Eléments finis*, *Etat adjoint*.

Participants : François Courty, Alain Dervieux.

Il s'agit dans ce nouveau sujet d'utiliser l'état adjoint pour rechercher le meilleur maillage pour la solution d'un écoulement. Il s'agira d'une extension de la méthode de la métrique continue [20].

9. Diffusion des résultats

9.1. 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 d'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, les avancées réalisées dans la mise en place de boucles d'optimisation avec état adjoint (notamment dans Aeroshape) sont 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 nous fournit des géométries d'avion d'affaire supersonique.

le projet bénéficie en outre des soutiens suivants :

- Projet « Instationnarités et structures à grande échelle dans des interactions onde de choc couche limite avec décollement » du Comité d'Orientation Supersonique du Ministère de la Recherche (COS).
- Contrat Dassault PEA (modéles réduits en Aérodynamque)
- Projet Liapunov « Turbulence and noise » avec IMM de Moscou.
- Projet Européen ADONIS de jumelage avec la Russie (Aéroacoustique)

9.2. Participation à des colloques, séminaires, invitations

- Présentations (F. Courty) aux réunions techniques du projet Aeroshape, du 29 au 31 janvier au CIRA, Capua, Italie, et les 19 et 20 juin à l'ONERA Chatillon.
- Invitation (L. Hascoët) à venir présenter les travaux du projet devant l'équipe d'Andreas Griewank à l'université de Dresde, du 28 au 30 janvier. Démonstration de l'outil TAPENADE.
- Présentation (L. Hascoët) de la DA et de TAPENADE à l'équipe de Charles Consel, projet COMPOSE à Bordeaux le 14 février, et de même dans le cadre d'une journée de cours organisée par le professeur M.Masmoudi à l'université de Toulouse, le 20 mars, et enfin dans le cadre de l'école CEA-EDF-INRIA « Problemes non-Linéaires Appliqués » à l'INRIA Rocquencourt du 25 au 28 mars.
- Conférence (F. Courty) aux Dixièmes Journées du groupe MODE (SMAI), en mars, « Application de la méthode one-shot aux algorithmes SQP en optimisation de formes ».
- Présentations (T. Kozubskaia) en mars, au PULV-Saint-Cloud et à l'université de Lyon.
- Participation (L. Hascoët) au workshop de Différentiation Automatique organisé du 24 au 26 avril à Aix-la-Chapelle.
- Conférence invitée (A. Dervieux) à JP60, le 12 juin à Jyvaskyla (Finlande).
- Conférence (M. Vazquez) à la conférence franco-germano-polonaise sur l'optimisation, à Cottbus du 9 au 13 septembre, « Gradient smoothness issues in supersonic shape design ».
- Conférence (F. Courty) à la conférence franco-germano-polonaise sur l'optimisation, à Cottbus du 9 au 13 septembre, « Trust region and one-shot approach to optimal control problems ».
- Conférence (F. Courty) au Workshop on Optimization in Partial Differential Equations and Applications, OPA 2002, Heidelberg, du 7 au 9 octobre, « Application of multilevel preconditioning to SQP algorithms ».

Durant l'année, le projet a invité Gilbert Strang (MIT), Paul Arminjon (université de Montréal), Charbel Farhat (université de Boulder Colorado), Eric Schall (université de Pau).

10. Bibliographie

Bibliographie de référence

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

[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.*. in « Numerical Methods in Engineering », 1996, pages 795-802, Proceedings of ECCOMAS'96.

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

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

[6] L. HASCOET, S. FIDANOVA, C. HELD. *Adjoining Independent Computations.* in « in "Automatic Differentiation of Algorithms, from Simulation to Optimization", Springer, LNCSE series », 2001, pages 299-304.

[7] L. HASCOËT. *A method for automatic placement of communications in SPMD parallelisation.* in « Parallel Computing Journal », numéro 27, 2001, pages 1655-1664.

[8] L. HASCOËT. *The Data-Dependence Graph of Adjoint Programs.* rapport de recherche, numéro 4167, INRIA, 2001, http://www.inria.fr/rrrt/rr-4167.html.

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

[10] M. TADJOUDDINE, C. FAURE, F. EYSSETTE. *Sparse Jacobian Computation in Automatic Differentiation by Static Program Analysis.* in « Static Analysis », série Lecture Notes in Computer Science, volume 1503, Springer-Verlag, éditeurs G. LEVI., pages 311-326, septembre, 1998.

Articles et chapitres de livre

[11] F. COURTY, A. DERVIEUX. *A One-shot trust-region algorithm for optimal control.* 2002, à paraître.

[12] A. DERVIEUX, F. COURTY, B. VÁZQUEZ. *Additive multilevel optimization and its application to sonic boom reduction.* in « Proceedings of Conference JP60, Jyvaskyla, 12-15 juin 2002 », 2002, à paraître.

[13] A. DERVIEUX, B. KOOBUS, C. FARHAT, R. LARDAT. *Application of unsteady fluid-structure methods to problems in aeronautics and space.* in « invited conference, CFSWA, Melbourne, december 2001 », 2002, à paraître.

[14] A. DERVIEUX, B. KOOBUS, C. FARHAT, M. VAZQUEZ, R. CARPENTIER, E. SCHALL. *Numerical models for computing unsteady fast flows and their interaction with structures.* in « invited conference at West-East High Speed Flow Fields, Marseille, april 2002 », 2002, à paraître.

[15] L. HASCOËT, R.-M. GREBORIO, V. PASCUAL. *Computing Adjoints by Automatic Differentiation with TAPENADE.* in « Ecole INRIA-CEA-EDF "Problèmes non-linéaires appliqués" », 2002, à paraître.

[16] L. HASCOËT, U. NAUMANN, V. PASCUAL. *TBR Analysis in Reverse Mode Automatic Differentiation.* in « Future Generation Computer Systems », 2002, à paraître.

[17] C. HELD, A. DERVIEUX. *One-shot airfoil optimisation without adjoint.* in « Computers and Fluids », numéro 31, 2002, pages 1015-1049.

Rapports de recherche et publications internes

[18] F. COURTY, A. DERVIEUX, B. KOOBUS, L. HASCOËT. *Reverse automated differentiation for optimum design : from adjoint state assembly to gradient computation.* rapport de recherche, numéro 4363, INRIA, 2002, http://www.inria.fr/rrrt/rr-4363.html.

[19] R. LARDAT, B. KOOBUS, F. RUFFINO, C. FARHAT, A. DERVIEUX. *Premières investigations du couplage fluide-structure autour d'un lanceur spatial générique.* rapport de recherche, numéro 4314, INRIA, 2002, http://www.inria.fr/rrrt/rr-4314.html.

[20] D. LESERVOISIER, P.-L. GEORGE, A. DERVIEUX. *Métrique continue et optimisation de maillage.* rapport de recherche, numéro 4172, INRIA, 2002, http://www.inria.fr/rrrt/rr-4172.html.

[21] M. VAZQUEZ, A. DERVIEUX, B. KOOBUS. *Aerodynamical and sonic boom optimization of a supersonic aircraft.* rapport de recherche, numéro 4520, INRIA, 2002, http://www.inria.fr/rrrt/rr-4520.html.

Bibliographie générale

[22] A. AHO, R. SETHI, J. ULLMAN. *Compilers : Principles, Techniques and Tools.* Addison-Wesley, 1986.

[23] J. ALLEN, K. KENNEDY. *Automatic translation of Fortran programs to vector form.* in « ACM Transactions on Programming Languages and Systems », numéro 4, volume 9, 1987, pages 491-542.

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

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

[26] C. BISCHOF, A. CARLE, P. KHADEMI, A. MAUER, P. HOVLAND. *ADIFOR2.0 User's Guide.* rapport technique, Argonne National Laboratory Technical Memorandum ANL/MCS-TM-192, and CRPC Technical Report CRPC-TR95516-S, 1998.

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

[28] I. CHARPENTIER. *Génération de codes adjoints : Traitement de la trajectoire du modèle direct.* rapport de recherche, numéro 3405, Inria, 1998, http://www.inria.fr/rrrt/rr-3405.html.

[29] I. CHARPENTIER, M. GHEMIRES. *Génération automatique de codes adjoints : Stratégies d'utilisation pour Odyssée, application au code meso-nh.* rapport de recherche, numéro 3251, Inria, 1997, http://www.inria.fr/rrrt/rr-3251.html.

[30] P. COUSOT. *Abstract Interpretation.* in « ACM Computing Surveys », numéro 1, volume 28, 1996, pages 324-328.

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

[32] P. FEAUTRIER. *Semantical analysis and mathematical programming ; application to parallelization and vectorization.* in « Workshop on Parallel and Distributed Algorithms », North Holland, éditeurs M. COSNARD, Y. ROBERT, P. QUINTON, M. RAYNAL., pages 309-320, Bonas, 1989.

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

[34] R. GIERING. *Tangent linear and Adjoint Model Compiler , Users manual 1.2.* 1997, http://www.autodiff.com/tamc.

[35] J. GILBERT, G. LEVEY, J. MASSE. *La différentiation automatique de fonctions representées par des programmes.* rapport de recherche, numéro 1557, Inria, 1991, http://www.inria.fr/rrrt/rr-1557.html.

[36] A. GRIEWANK. *Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation.* in « Optimization Methods and Software », volume 1, 1992, pages 35-54.

[37] A. GRIEWANK, D. JUEDES, J. UTKE. *Adol-C : a package for the automatic differentiation of algorithms written in C/C++.* in « ACM Transactions on Mathematical Software », volume 22, 1996, pages 131-167.

[38] 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 », SIAM, éditeurs M. BERZ, C. BISCHOF, G. CORLISS, A. GRIEWANK., pages 95-106, 1996, http://www.inria.fr/rrrt/rr-2794.html, rapport de Recherche INRIA 2794.

[39] M. IRI. *History of automatic differentiation and rounding estimation.* in « Automatic Differentiation of Algorithms : Theory, Implementation and Application », SIAM, éditeurs A. GRIEWANK, G. CORLISS., pages 1-16, 1991.

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

[41] K. KUBOTA. *PADRE2 version 2α, user's manual.* 1996.

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

[43] F. LEDIMET, O. TALAGRAND. *Variational algorithms for analysis and assimilation of meteorological observations : theoretical aspects.* in « Tellus », volume 38A, 1986, pages 97-110.

[44] K. PINGALI, M. BECK, R. JOHNSON, M. MOUDGILL, R. STODGHILL. *Dependence Flow Graphs : an algebraic approach to program dependencies.* Pitman, 1991.

[45] T. REPS, S. HORWITZ, W. YANG. *Detecting program components with equivalent behaviors.* rapport technique, numéro 840, Computer Sciences Lab., University of Wisconsin, Madison, 1989.

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

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

[48] N. ROSTAING. *Différentiation Automatique : application à un problème d'optimisation en météorologie.* thèse de doctorat, université de Nice Sophia-Antipolis, 1993.

[49] O. TALAGRAND. *The use of adjoint equations in numerical modelling of the atmospheric circulation.* in « Automatic Differentiation of Algorithms : Theory, Implementation and Application », SIAM, éditeurs A. GRIEWANK, G. CORLISS., pages 169-180, 1991.

[50] C. TEAM. *Centaur 2.0 Documentation.* Inria, 1994.

[51] H. ZIMA, B. CHAPMAN. *Supercompilers for Parallel and Vector Computers.* ACM Press, 1990.