INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Avant-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
Ina Castrogiovanni [TR, à temps partiel dans le projet]

Personnel Inria
Christèle Faure [CR contractuel jusqu'au 31/8/2000]

Professeur en Sabbatique
Andreas Griewank [Université de DRESDE (Allemagne), jusqu'au 31/8/1999]

Ingénieurs experts
Patrick Dutto [Contrat Génie, jusqu'au 31/10/1999]
Stefka Fidanova [Contrat Décision]
Frédéric Olier [Contrat EDF]

Chercheurs doctorants
Mohamed Tadjouddine [jusqu'au 15/3/1999]

Chercheur post-doctorant
Uwe Naumann [Post Doctorant MIO]

2 Présentation et objectifs généraux

L'activité du projet TROPICS concerne les outils logiciels pour l'analyse et la transformation semi-automatique des programmes de calcul scientifique. Cette activité inclut la Différentiation Automatique, avec le logiciel ODYSSÉE, qui était développé précédemment dans le projet SAFIR. L'activité inclut aussi la parallélisation SPMD, étudiée auparavant dans le cadre du projet SINUS (logiciel GASPARD). On peut décomposer l'activité de la manière suivante :

- La Différentiation Automatique : optimisations mémoire pour le mode inverse (adjoints) et les calculs de Jacobienne, 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,

– 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.

Le projet est a priori candidat pour étudier et développer d'autres outils qui faciliteraient la tâche des développeurs en calcul scientifique.

3 Fondements scientifiques

3.1 Différentiation Automatique

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

Participants : Patrick Dutto, Christèle Faure, Stefka Fidanova, Andreas Griewank, Laurent Hascoët, Uwe Naumann, Frédéric Olier, Valérie Pascual, Mohamed Tadjouddine.

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.

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,

– Par transformation de source à source (outils ODYSSÉE [2], ADIFOR [BCK+98], TAMC [Gie97], PADRE2 [Kub96]); cette approche demande le développement d'outils complexes, mais

[GLM91] J. Gilbert, G. LeVey, J. Masse, « La differentiation automatique de fonctions representées par des programmes », *Rapport de recherche n° 1557*, Inria, 1991.

[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.

[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.

permet une plus grande flexibilité.

On trouvera une liste complète des systèmes actuels de D.A. à l'adresse : `http://www-sop.inria.fr/safir/SAM/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

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

  . 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,

[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.

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.

3.2 Parallélisation

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

Participants : Patrick Dutto, Laurent Hascoët, Stéphane Lanteri [projet SINUS].

[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.

[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.

[Cha98] I. Charpentier, « Génération de codes adjoints : Traitement de la trajectoire du modèle direct », *Rapport de recherche n° 3405*, Inria, 1998.

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.

Le domaine de la parallélisation est trop vaste pour espérer en faire ici un tableau complet. Il est aussi en évolution constante, rapide, suivant l'architecture des machines. 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.

Dans ce contexte, nos directions de recherche actuelles sont, d'une part, la parallélisation semi-automatique d'instructions, en direction d'OpenMP [Ope99] et, d'autre part, l'aide à la parallélisation SPMD dans le cas des boucles itératives hautement parallèles des calculs sur les maillages.

La parallélisation des instructions en direction d'OpenMP constitue en quelque sorte un prolongement moderne de la vectorisation. Les questions d'accès à la mémoire sont laissées au compilateur, et il ne reste qu'à trouver des ensembles d'instructions indépendantes, que le langage cible permet d'exploiter en termes de constructeurs syntaxiques parallèles. Les questions habituelles de découpage des boucles et d'expansion des variables se retrouvent, parfois sous un autre nom (e.g. « localisation »). Les questions concernent l'exploitation optimale des possibilités d'expression nouvelles d'OpenMP. Par ailleurs, la parallélisation multi-processeurs par échange de messages étant dorénavant incontournable, une réflexion intéressante concerne la collaboration entre OpenMP et MPI, par exemple à deux niveaux de granularité différents dans un même programme.

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

[Ope99] OpenMP Architecture Review Board, *OpenMP, Simple, Portable, Scalable SMP Programming*, 1999, `http://www.openmp.org/`.

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 [3] [11], 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 : Valérie Pascual, Laurent Hascoët, Frédéric Olier.

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, etc...

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 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 dépendance*. 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 cherche à définir un langage impératif abstrait, vers lequel on peut projeter chaque langage impératif réel. La plate-forme ne connaîtra que ce langage abstrait, ce qui facilitera grandement le passage d'un langage à l'autre.

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. La transformation qui nous occupe le plus est la Différentiation Automatique, suivie de la parallélisation. D'une manière générale, les dérivées d'un programmes sont utiles pour plusieurs types d'applications :

- la simulation de systèmes complexes,
- les études de sensibilité,
- l'analyse des propagations d'erreurs d'arrondi,
- 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,
- 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.

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.

[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.

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. Dans ce cas, 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.

4.3 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].

4.4 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

[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.

inverse de la Différentiation Automatique. Cette approche a déjà été validée, par exemple dans la thèse de Nicole Rostaing [Ros93]

5 Logiciels

5.1 Odyssée

Participants : Christèle Faure [correspondante], Frédéric Eyssette [UNSA], André Galligo [UNSA], Vladimir Vyskocil [UNSA], Stéphane Dalmas [projet CAFE], José Grimm [projet MIAOU], Mohamed Tadjouddine, Uwe Naumann, Frédéric Olier.

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 et permet de générer en mode direct un code linéaire tangent et, en mode inverse, un linéaire cotangent. Ce système est diffusé aux industriels dans le cadre de contrats d'études : EDF, Dassault, Elf, Essilor, CNES, ... et dans le secteur académique par mise à disposition gracieuse. Pour donner des informations au jour le jour sur ce système, deux pages sont maintenues (une en français et une en anglais) à : `http://www.inria.fr/safir/SAM/Odyssee/odyssee.html`

La version `1.7` d'ODYSSÉE a été déposée à l'APP en septembre 1999 conjointement avec l'Université de Nice-Sophia Antipolis.

Le nombre de mises à disposition effectuées depuis janvier 1999 est de :

- 4 à l'étranger,
- 4 en France,
- 3 projets de l'INRIA.

5.2 Gaspard

Participants : Laurent Hascoët [correspondant], Stéphane Lanteri.

GASPARD est un logiciel d'aide au placement des routines de communications. Il est spécialisé pour la parallélisation SPMD des programmes de résolution itérative sur maillages non structurés, par une approche de partition du maillage. A partir du programme séquentiel multi-procédural, et d'une description du type de *recouvrement* entre les sous-maillages, GASPARD indique à l'utilisateur les emplacements du programme où il est nécessaire d'insérer des appels à des routines de communication, pour assurer les mises à jour entre les sous-maillages. Ces emplacements sont calculés par une analyse du *graphe de dépendance* du programme.

GASPARD a été développé dans le cadre de la collaboration GÉNIE, entre DASSAULT, AÉROSPATIALE, et l'INRIA. Ce logiciel a été construit en Le-Lisp, sur la plate-forme FORESYS-PARTITA, de SIMULOG.

[Ros93] N. ROSTAING, *Différentiation Automatique: application à un problème d'optimisation en météorologie*, thèse de doctorat, Université de Nice Sophia-Antipolis, 1993.

5.3 ALI

Participants : Laurent Hascoët [correspondant], Valérie Pascual, Frédéric Olier, Uwe Naumann.

ALI est le nom donné à la plate-forme commune décrite dans la section 3.3. Le développement d'ALI a démarré en 1999. D'une part, ALI est utilisé pour implémenter les analyses de D.A. plus efficaces décrites dans la section 7.1. D'autre part, nous allons proposer un projet européen dans lequel ALI sera la base de développement. ALI est une plate-forme pour l'*Analyse des Langages Impératifs*. ALI permet d'analyser des programmes, qui sont représentés dans un langage interne abstrait, nommé IL, indépendant 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 analyse chaque sous-programme sous la forme d'un graphe de flot de contrôle, associé à une table de symboles. Il implémente les analyses utiles pour la plupart des transformations de programme, et en particulier pour la Différentiation Automatique.

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.

6 Résultats nouveaux

6.1 Différentiation Automatique hiérarchique

Participante : Christèle Faure.

Mots clés : différentiation automatique, transformation de code, code adjoint, dérivée, compromis stockage/recalcul.

Résumé : *La complexité en temps de la dérivée en mode inverse d'un programme général peut être déduite directement des travaux sur la complexité en nombre d'opérations d'un programme sans boucle, si on exclut du temps d'exécution les accès à la trajectoire. Or, lors de l'exécution des codes adjoints (mode inverse), le coût en temps de ces accès est non négligeable. Nous avons donc étudié ces coûts en fonction de l'algorithme de dérivation utilisé.*

Des travaux sur la complexité en nombre d'opérations arithmétiques (voir [BS83,Mor85]) ont été réalisés en prenant comme modèle de programme des programmes sans boucles. Nous avons étendu ces résultats à une hiérarchie d'unités de dérivation (instructions, blocs d'instructions ou unités de compilation). Pour cela, nous avons décrit les différents algorithmes de dérivation en mode inverse. Dans cette étude, les algorithmes utilisés lors de l'écriture d'adjoints à la main ont été considérés autant que ceux utilisés lors de la génération automatique. Un panorama complet

[BS83] W. Baur, V. Strassen, « The complexity of partial derivatives », *Theoretical Comp. Sci. 22*, 1983, p. 317–330.

[Mor85] J. Morgenstern, « How to compute fast a function and all its derivatives, a variation on the theorem of Baur-Strassen », *Sigact News 16*, 1985, p. 60–62.

des méthodes a ainsi été obtenu, et une comparaison de la complexité de leur application à une hiérarchie d'unités de dérivation a pu être réalisée (voir [10]). Les compromis stockage/recalcul sous-jacents à chaque méthode ont pu eux aussi être décrits et évalués sur THYC1D (voir [7]).

6.2 Différentiation Automatique des boucles parallélisables

Mots clés : différentiation automatique, mode inverse, mode adjoint, parallélisme.

Participants : Stefka Fidanova, Christophe Held [projet SINUS], Laurent Hascoët, Alain Dervieux [projet SINUS].

Le domaine d'application de la D.A. est le calcul scientifique. Ici, les programmes sont très coûteux, et leur différentiation en mode inverse délicate. On est à la recherche de stratégies permettant de réduire le nombre de valeurs à sauvegarder pendant la phase « progressive » du calcul. De plus, le programme initial est très souvent parallélisé, par échange de messages entre processus parallèles.

Le noyau calculatoire de ces programmes est constitué de boucles qui sont fréquemment du type « à itérations indépendantes », c'est à dire parallélisables. Pour ces boucles, éventuellement terminées par une opération de réduction additive, on peut montrer que la différentiation en mode inverse *commute* avec le contrôle de la boucle. Cela signifie que le code différentié peut être écrit comme une boucle, dont chaque itération est d'abord progressive, puis inverse. L'avantage est évidemment une diminution radicale des sauvegardes, puisqu'on ne sauvegarde qu'une itération à la fois. On a montré la faisabilité de cette méthode particulière, dans le cas d'une boucle isolée, puis dans le cas général de boucles parallèles incluses dans un programme. Les résultats sont très intéressants, puisque, pour chaque boucle, la taille des sauvegardes diminue d'un facteur égal au nombre d'itérations de ces boucles, c'est-à-dire en fait au nombre d'éléments du maillage. En contrepartie, chacune de ces boucles différentiées doit disposer, lors de l'exécution, des variables adjointes provenant de la suite du programme, ce qui requiert d'utiliser un mécanisme de type « checkpointing » avant ces boucles.

En collaboration avec le projet SINUS, on veut démontrer l'applicabilité de cette tactique dans un cas précis. On considère la partie « calcul des flux » dans un programme de résolution des équations d'Euler. Le gradient de ces flux est ensuite calculé, pour être utilisé enfin dans la résolution d'un système linéaire. On se propose de remplacer ce calcul de gradient, jusqu'ici écrit avec le reste du programme, par un module obtenu par Différentaition Automatique en mode inverse. Cela deviendrait praticable grâce à la tactique précédente, qui éviterait la consommation mémoire énorme d'un code inverse ordinaire.

6.3 Mode Direct Adjoint

Mots clés : différentiation automatique, mode direct, mode inverse, mode adjoint.

Participants : Andreas Griewank, Christèle Faure, Vittorio Selmin [Alenia, Italie].

L'application du mode inverse à des boucles itératives demande, en première approximation, le stockage de tous les états intermédiaires. Si le nombre total d'itérations est très grand, on peut

utiliser une sorte de « checkpointing » [Gri92]. Cela réduit considérablement la consommation mémoire. Néanmoins, cette technique entraîne des recalculs. Nous avons trouvé que tout cela n'est pas nécessaire si le programme itératif calcule un point fixe, au sens mathématique. Nous avons proposé un mode inverse qui n'a besoin que du stockage des valeurs locales de chaque itération. Ce mode produit des gradients approchés qui convergent à la même vitesse que le point fixe lui-même. Cette approche a été vérifiée sur un code Navier-Stokes fourni par Alenia. Nous avons de plus développé une combinaison des modes direct et inverse qui aboutit à une forme de superconvergence.

6.4 Optimisation du calcul de la Jacobienne

Mots clés : différentiation automatique, Jacobienne, analyse des dépendances, régions de tableaux.

Participants : Mohamed Tadjouddine.

Cette année, Mohamed Tadjouddine a achevé sa thèse [5], qui concerne les optimisations pour un calcul efficace de la matrice jacobienne. Le mode de Différentiation Automatique qui calcule la matrice jacobienne est utile, par exemple pour la *simulation*. Un exemple a été fourni par le contrat européen AD-CAPE, qui s'est terminé cette année, et qui utilisait ODYSSÉE.

Malheureusement, ce calcul est coûteux, à cause de la taille de la matrice jacobienne finale, et surtout à cause de la taille des matrices jacobiennes partielles, au cours du calcul. On observe cependant que ces matrices sont souvent très creuses. Une partie de cette structure creuse peut être détectée statiquement, principalement dans les parties suffisamment régulières du programme. Une autre partie de cette structure creuse ne pourra être détectée et utilisée que dynamiquement. C'est le but de l'utilisation de la bibliothèque SparsLinC par ADIFOR. Ces deux approches sont complémentaires. Mohamed Tadjouddine a orienté ses efforts sur la détection et l'utilisation de la structure creuse par des analyses statiques.

Cette détection est possible par une analyse du programme qui utilise le graphe de dépendance. Le travail est donc de détecter la structure creuse de la matrice en analysant les indices affines des tableaux à l'intérieur des boucles régulières, puis de choisir une représentation de cette structure creuse, et enfin d'exploiter cette information dans le code dérivé. Une question délicate est d'assurer une cohérence entre ces phases. Il est par exemple inutile de détecter une structure creuse extrêmement complexe, dont l'exploitation dans le code dérivé serait trop lourde.

Mohamed Tadjouddine a proposé dans sa thèse un certain degré de précision des analyses, analogue à ce qui se pratique dans les analyses de régions de tableau. Voir par exemple [CI96], ou [Cre96]. Il a ensuite décrit son implémentation des analyses dans PARTITA, puis l'utilisation

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

[CI96] B. CREUSILLET, F. IRIGOIN, « Exacs vs approximate array region analyses », Languages and compilers for parallel computing.

[Cre96] B. CREUSILLET, *Analyses de régions de tableaux et applications*, thèse de doctorat, Ecole des mines de Paris, 1996.

qui en est faite dans ODYSSÉE. Ce travail montre également qu'il reste des pistes pour un calcul plus efficace des matrices jacobiennes.

6.5 Langage Impératif Abstrait

Mots clés : langage impératif, syntaxe abstraite, langage intermédiaire.

Participants : Valérie Pascual, Frédéric Olier, Laurent Hascoët.

Pour construire la plate-forme d'analyse de langages impératifs ALI, on a dû définir un langage interne, abstrait (sans syntaxe concrète), dans lequel on puisse traduire sans perte sémantique des programmes écrits dans l'un des principaux langages impératifs, tels que FORTRAN, FORTRAN90, C, C++,... Il était important que ce langage abstrait ne soit pas la juxtaposition des langages réels individuels. Il fallait en particulier que les constructeurs similaires correspondent tous à un seul constructeur abstrait. Il fallait ensuite que ces constructeurs abstraits se prêtent bien aux analyses ultérieures, et enfin que la traduction inverse vers le langage initial soit possible.

Nous avons défini une première version de ce langage abstrait, baptisé IL. Nous avons expérimenté des outils de traduction, depuis les syntaxes abstraites de FORTRAN ou C, vers IL. La représentation interne des programmes dans ALI s'appuie exclusivement sur IL.

6.6 Parallélisation

Mots clés : parallélisation, SPMD, partition de maillage, élément fini, Navier-Stokes, OpenMP, transformation de boucles.

Participants : Laurent Hascoët, Stéphane Lanteri.

De nouvelles expériences d'utilisation de GASPARD ont été réalisées. Un rapport de recherche présentant l'ensemble des études et développements autour de GASPARD a été rédigé [11]. L'insertion des communications par l'outil s'appuie sur la propagation, le long des flèches du graphe de dépendance, d'une information signifiant qu'une valeur est à jour sur le recouvrement ou non. Les évolutions autorisées de ce statut (« à jour » ou pas), sont spécifiées par un automate fini caractéristique du recouvrement choisi. Ces automates sont pour l'instant prédéfinis. Une question était de savoir si ces automates peuvent se calculer. On a proposé une manière de déduire tous ces automates d'un ensemble fini (et petit) de transitions élémentaires, qui découlent de propriétés géométriques des tétraèdres, triangles, segments, et sommets des maillages.

Par ailleurs, dans le cadre d'une collaboration avec Simulog, Laurent Hascoët a étendu l'outil de parallélisation semi-automatique PARTITA, pour une génération de code parallélisé OpemMP. Cela a permis de rationaliser le mécanisme de paramétrage du langage parallèle cible dans PARTITA. Le peu de temps consacré à cette extension montre à notre avis la pertinence de ce paramétrage, qui a permis de recycler facilement des analyses déjà implémentées pour d'autres langages cibles (HPF...), en direction d'OpenMP. Par exemple, la détection des boucles `PARALLEL DO` existait déjà, ainsi que celle des variables *localisées* de ces boucles, parce que ces traits étaient déjà exploités dans la traduction vers HPF. Il a suffi de changer quelques

paramètres pour générer la syntaxe OpenMP visée. Le même mécanisme s'est appliqué pour la détection des opérations de réduction, ou pour celle des barrières de synchronisation.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 EDF

Participants : Frédéric Olier, Christèle Faure, Laurent Hascoët.

La différentiation de très gros programmes par Odyssée pose des problèmes d'analyse statique du code original: le temps d'analyse est trop important. Ce problème n'était pas apparu lors de nos études précédentes sur THYC3D, et a été identifié lors d'un nouveau contrat sur un code appelé COCCINELLE.

La première partie de l'étude actuelle avait pour but de déterminer la cause de ce surcoût et a conclu à un défaut d'implémentation des informations propagées lors de l'analyse. L'utilisation de listes de chaînes de caractères qu'il faut parcourir, fusionner ... semble être à l'origine du problème. Le fait que ce problème soit apparu sur COCCINELLE et non sur THYC3D reste cependant inexpliqué car les tailles de ces codes en nombre de lignes et de variables sont comparables.

La deuxième partie de l'étude actuelle avait pour but de définir les éléments de la syntaxe Fortran 77 non traités ou traités incorrectement par Odyssée. Cette partie a montré qu'en dehors des limitations décrites dans le manuel d'utilisation, le mécanisme de transmission d'information à l'utilisateur (sous forme de warning ou d'erreurs) n'était pas systématiquement utilisé. Il serait simple de transmettre ces messages en écrivant systématiquement un fichier de log. Pour l'instant, l'utilisation d'un fichier d'erreur est à la charge de l'utilisateur. Quant aux limitations d'Odyssée, leur levée concernant les variables rémanentes (mal traitées) est facile à mettre en place par un prétraitement. Le traitement des `EQUIVALENCE`s (non traitées) et des espaces communs (une définition unique dans le programme lu), pourrait être réalisé en utilisant une cartographie de la mémoire au lieu des noms de variable.

Ces deux phases ont montré la nécessité de rendre plus efficace l'analyse du code, mais pas celle de changer la dérivation proprement dite. Le choix retenu est de déléguer à ali les phases d'analyse incriminées. Cette solution nécessite l'interfaçage d'Odyssée et d'ali, ce qui est en cours d'implémentation.

7.2 Actions nationales

7.2.1 Action de Recherche Coopérative MIO

Mots clés : différentiation automatique, mode inverse, mode adjoint.

Participants : Christèle Faure, Isabelle Charpentier, Alain Deutsch, Andreas Griewank, Jean-Charles Gilbert, Laurent Hascoët, François Xavier Le Dimet, Uwe Naumann, Yves Papegay.

L'action de recherche coopérative baptisée MIO (Mode Inverse Opérationnel) a débuté en

juin 1997. Elle a pour but de développer de nouveaux algorithmes de Différentiation Automatique en mode inverse et de les implémenter dans Odyssée.

Yannick Tremolet est venu du 8 au 12 mars et nous a présenté ses travaux au National Center for Environmental Predictions (USA). Il s'agissait de décrire l'utilisation d'Odyssée pour dériver un code opérationnel utilisant MPI pour transférer les données.

De plus, deux personnes sont venues soutenir nos efforts de recherche : Andreas Griewank comme professeur invité et Uwe Naumann comme étudiant post-doctoral. Le point particulier qui nous a intéressés cette année a été la suppression des sauvegardes inutiles dans la partie progressive du code généré en mode inverse. Ce premier point a été choisi comme thème principal à partir des résultats obtenus sur Meso-NH [Cha99] d'une part et sur THYC1D [7] d'autre part. En effet, ces travaux ont montré qu'Odyssée générait des sauvegardes inutiles. Nous avons identifié trois causes de consommation abusive de sauvegardes :

1. Certaines valeurs ne sont utilisées qu'à l'intérieur d'expressions linéaires. On sait alors que les dérivées de ces expressions ne feront pas intervenir cette valeur, mais seulement sa dérivée. Il est inutile de sauvegarder cette valeur.
2. Les appels de sous-programmes sont considérés comme des instructions atomiques à l'intérieur du sous-programme appelant. Ces sous-programmes sont susceptibles d'écraser un certain nombre de valeurs, qu'il faut donc souvent sauvegarder (au point précédent près). Quand ces valeurs sont des éléments d'un tableau, on sauvegarde souvent l'ensemble du tableau, ce qui est abusif.
3. Lorsqu'on veut implémenter un compromis stockage-recalcul au niveau d'un appel de sous-programme ou d'une itération de boucle, on doit sauver les valeurs nécessaires pour relancer le calcul en ce point. L'approche naïve consiste à sauver les valeurs de toutes les variables. Il serait plus économique de ne sauvegarder que les variables effectivement utiles dans la suite. Ces variables peuvent être détectées par une analyse globale.

Cette année, nous avons défini les propriétés qui caractérisent les variables correspondant aux cas (1) et (3). Ces propriétés peuvent être calculées par une analyse statique du programme initial. Ces analyses consistent en la propagation d'une information abstraite à travers le programme, représenté soit sous la forme d'arbre de syntaxe abstraite, soit sous la forme de graphe de flot de contrôle. Nous avons défini la forme de ces informations et leurs règles de propagation„ c'est-à-dire en termes d'interprétation abstraite le *domaine sémantique* et les opérations abstraites sur ce domaine. Après étude de la terminaison de ces algorithmes, ceci a donné lieu à une première implémentation dans ODYSSÉE.

En réponse au cas (2), nous avons défini une stratégie de sauvegarde différente pour le mode inverse, nommée « trajectoire globale ». Chaque sous-programme est dorénavant responsable de la restauration des valeurs des variables qu'il modifie. Par conséquent, le programme appelant n'a plus à sauvegarder les tableaux éventuellement modifiés par le sous-programme. Cette stratégie conduit à une diminution très nette des sauvegardes.

Une première évaluation des résultats a été effectuée sur THYC1D : le nombre de sauvegardes est divisé par trois en utilisant ces tactiques. On peut remarquer (sur cet exemple) que

[Cha99] I. CHARPENTIER, « The MesODiF package for gradients computations with the atmospheric model Meso-NH », *Environemental modeling and software*, 1999, Air Pollution Modelling and Simulation 98 (à paraître).

la diminution la plus importante a été obtenue sur les variables booléennes. La validation de cet algorithme n'a pu être menée à bout cette année, mais nous espérons vérifier l'ordre de grandeur de ce gain sur plusieurs exemples dont le code industriel THYC3D.

L'action incitative MIO se termine en décembre 1999. Elle a permis une réflexion commune entre les différentes équipes de l'INRIA impliquées. Ce travail n'a pas produit directement des améliorations dans Odyssée, mais a permis de faire avancer la connaissance des problèmes posés par le mode inverse. On peut conclure que les efforts à produire pour rendre le mode inverse directement applicable à des codes de taille industrielle sont de deux ordres : d'une part, il faut continuer à étudier les algorithmes d'analyse statique (travail commencé avec le projet PARA) nécessaires à la dérivation et, d'autre part, et de façon plus fondamentale, il faut réfléchir aux algorithmes de dérivation en mode inverse (voir [10]). Un travail de décomposition des algorithmes de DA en fonctionnalités élémentaires commencé en collaboration avec Andreas Griewank doit se poursuivre pour obtenir une compréhension totale des qualités des méthodes utilisées et en définir de nouvelles. L'action MIO a aussi permis de créer une synergie centrée sur l'INRIA entre les équipes de Différentiation Automatique européennes. Par exemple, elle a permis la préparation du « 3rd International Conference/Workshop on Automatic Differentiation : From Simulation to Optimization » qui sera organisé à Nice (19-23 Juin 2000).

7.3 Actions européennes

7.3.1 DECISION

Mots clés : différentiation automatique, mode direct, mode inverse, parallélisme, SPMD, MPI.

Participants : Christèle Faure, Patrick Dutto, Stefka Fidanova.

Nous participons au projet européen DECISION (Integrated Optimisation Strategies for increased Engineering Design Complexity) qui a débuté en octobre 1997 et qui dure 30 mois. C'est un projet HPCN du domaine 6 : Simulation and Design.

Le projet intervient dans l'étude des liens existant entre Différentiation Automatique et parallélisme : dériver un code parallèle. Nous avons choisi de considérer la librairie MPI comme une librairie externe et de ne pas modifier le noyau principal d'Odyssée. Cette extension consiste en un fichier qui, lu avant la dérivation, permet à Odyssée de pratiquer une analyse de dépendance complète, et en plusieurs fichiers Fortran 77 contenant les définitions des dérivées des routines de communication, qui sont compilées et linkées au code généré pour exécuter la dérivée. La validation de cette extension faite sur le code NS3D (fourni par le projet SINUS) a été faite en deux étapes. La première (voir [8]) pour le mode direct de différentiation automatique, n'a pas présenté de problème. La deuxième, qui concerne le mode inverse (voir [9]), a montré la difficulté de transposer certaines transformations telles que les opérations de décomposition (split) et recomposition des données (gather). Ces travaux ont donné lieu à une publication [6] à la conférence ENUMATH'99 .

7.4 Participation à des colloques, séminaires, invitations

Christèle Faure a présenté ses travaux sur les adjoints générés automatiquement dans diverses réunions nationales et internationales :

- « Journées scientifiques sur l'assimilation d'observations de la chimie atmosphérique dans les modèles », Paris, 28-29 Janvier 1999,
- M18 review meeting du projet DECISION à Oxford, Angleterre, 22-23 avril 1999,
- « ERCOFTAC Workshop on Adjoint Methods », IMFP, TOULOUSE, 21-23 Juin 1999,
- « Journée sensibilisation aux problèmes adjoints », EDF Chatou, 22 octobre 1999.

Christèle Faure a aussi présenté une contribution à :

- ENUMATH'99 : Third European Conference on Numerical Mathematics and Advanced Applications, University of Jyvaskila, Finland, July 26-30, 1999.

Laurent Hascoët a présenté les travaux du projet :

- au « ERCOFTAC Optimum Design Symposium », à Munich, Allemagne, les 17 et 18 mai 1999,
- au LIAMA à Beijing, Chine, du 9 au 13 août 1999. Cette visite a été financée par le projet LIAMA « Hierarchical Methods for solution and optimisation of nonlinear PDEs » (Jean-Antoine Désidéri, Xu Xue-jun).

8 Diffusion de résultats

8.1 Animation de la communauté scientifique

Le projet organise, en coopération avec Andreas Griewank, la 3$^{\text{e}}$ conférence internationale sur la Différentiation Automatique, qui se tiendra à Nice du 19 au 23 juin 2000, `http://www-sop.inria.fr/tropics/ad2000/`.

Laurent Hascoët a participé au comité d'organisation de la conférence « Problem-Solving Environments: Infrastructure and Prototypes », organisée par l'European Science Foundation, à San Feliu de Guixols, Espagne, du 12 au 17 juin 1999.

9 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] 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.

[2] C. Faure, Y. Papegay, « Odyssée User's Guide Version 1.7 », *Rapport technique n^{o} 224*, INRIA, 1998.

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

[4] 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.

Thèses et habilitations à diriger des recherches

[5] M. Tadjouddine, *Analyse de Dépendances de Jacobiennes Creuses pour la Différentiation Automatique*, thèse de doctorat, Université de Nice-Sophia Antipolis, 1999.

Communications à des congrès, colloques, etc.

[6] P. Dutto, C. Faure, S. Fidanova, « Automatic differentiation and parallelism », *in : Proceedings of Enumath 99, Finland*, 1999.

Rapports de recherche et publications internes

[7] C. Faure, I. Charpentier, « Comparing automatically generated and hand coded adjoints », *Rapport de recherche n° 3811*, INRIA, 1999.

[8] C. Faure, P. Dutto, « Extension of Odyssée to the MPI Library -Direct mode- », *Rapport de recherche n° 3715*, INRIA, juin 1999, `http://www.inria.fr/RRRT/RR-3715.html`.

[9] C. Faure, P. Dutto, « Extension of Odyssée to the MPI Library -Reverse mode- », *Rapport de recherche n° 3774*, INRIA, 1999.

[10] C. Faure, « Adjoining strategies for multi-layered programs », *Rapport de recherche n° 3781*, INRIA, 1999.

[11] L. Hascoët, « Parallelization of Finite Element Codes with Automatic Placement of Communications », *Rapport de recherche n° 3646*, Inria, 1999.

[12] U. Naumann, « Efficient Calculation of Jacobians Using Dynamic Programming », *Rapport de recherche n° 3689*, Inria, 1999.

[13] U. Naumann, « Heuristics for Efficiently Calculating Jacobians by Edge Elimination in Computational Graphs », *Rapport de recherche n° 3690*, Inria, 1999.

[14] U. Naumann, « Optimizing the Accumulation of Jacobians by Edge Elimination in the Computational Graph », *Rapport de recherche n° 3659*, Inria, 1999.

[15] U. Naumann, « SAVE - Simulated Annealing Applied to the Vertex Elimination Problem in Computational Graphs », *Rapport de recherche n° 3660*, Inria, 1999.