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

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

Personnel Inria

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

Ingénieurs experts

Stefka Fidanova [Contrat Décision jusqu'au 31/1/2000]

Frédéric Olier [Contrat EDF jusqu'au 30/9/2000]

Stagiaires

Martial Bonaventure [Elève Ingénieur INSA Rouen, du 1/6/2000 au 31/8/2000]

Jean-Philippe Sautarel [Elève Ingénieur INSA Rouen, du 1/6/2000 au 31/8/2000]

2 Présentation et objectifs généraux

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

Les axes de recherche du projet sont :

- La Différentiation Automatique : optimisations mémoire pour le mode inverse (adjoints) et les calculs de Jacobiennes, différentiation de programmes parallèles, différentiation adaptée de sous-programmes particuliers,

- La Parallélisation SPMD (*Single Program, Multiple Data*), appliquée aux programmes itératifs sur maillages irréguliers : détection et minimisation des besoins de communication entre processeurs,
- La comparaison de programmes, capable de détecter une équivalence plus sémantique que syntaxique.
- L'outillage commun aux logiciels de transformation de programmes scientifiques : représentation interne de gros programmes en langages impératifs, graphes d'appel, graphes de flot, graphes de dépendances.

Nous maintenons des relations avec l'université de Dresde (Allemagne) et l'université de Hatfield (GB). ODYSSÉE est distribué librement, et PARTITA est commercialisé par SIMULOG.

3 Fondements scientifiques

3.1 Différentiation Automatique

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

Participants : Martial Bonaventure, Christèle Faure, Stefka Fidanova, Laurent Hascoët, Frédéric Olier, Valérie Pascual, Jean-Philippe Sautarel.

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,

[GLM91] J. GILBERT, G. LEVEY, J. MASSE, «La différentiation 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.

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

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

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

– Le *mode « direct »* construit le programme

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

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

– Le *mode « matrice jacobienne »* construit le programme

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

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

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

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

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

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

– 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 [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.

[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.

- 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, Stefka Fidanova.

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

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

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 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 [5] [6], 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 : Martial Bonaventure, Laurent Hascoët, Frédéric Olier, Valérie Pascual, Jean-Philippe Sautarel.

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 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

[ASU86] A. AHO, R. SETHI, J. ULLMAN, *Compilers: Principles, Techniques and Tools*, Addison-Wesley, 1986.

[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.

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

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

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 programme sont utiles pour plusieurs types d'applications :

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

[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: 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.

- 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.

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].

[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.

[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.

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 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], André Galligo [UNSA], 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. ODYSSÉE génère un code différentié soit en mode direct (« linéaire tangent »), soit en mode inverse (« linéaire cotangent »). La version `1.7` d'ODYSSÉE Odyssée est maintenant en accès libre, avec le source Caml, sur le serveur FTP du projet `ftp://ftp-sop.inria.fr/tropics`.

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. À 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, Martial Bonaventure, Jean-Philippe Sautarel.

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, ODYSSÉE sous-traite désormais à ALI la détection des dépendances numériques, qui est la phase la plus coûteuse de la différentiation automatique. 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++... 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 API (Application Programmer Interface) est en cours de définition, pour permettre à un développeur d'application de construire des analyses et transformations spécifiques à partir des structures fournies par ALI.

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.

6 Résultats nouveaux

6.1 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].

Les boucles parallélisables sont très fréquentes en calcul scientifique. Nous proposons une manière de tirer parti de cette propriété pour réduire la consommation mémoire des adjoints de ces boucles parallèles.

Nous considérons les boucles dont les différentes itérations ou bien sont purement indépendantes, ou bien accumulent des valeurs en les additionnant dans une variable partagée (« réductions additives »). Nous avons montré que la boucle adjointe correspondante est elle aussi constituée d'itérations indépendantes ou de réductions additives. À partir de cela, nous pouvons prouver que la différentiation en mode inverse *commute* avec le contrôle de la boucle. On en tire une méthode de différentiation adaptée, où chaque itération est immédiatement suivie de son adjoint. Cela donne un code adjoint équivalent, qui consomme beaucoup moins de mémoire pour conserver les valeurs intermédiaires du calcul.

Une première démonstration de la faisabilité de la méthode a été faite. Le code fourni par le projet Sinus a été différentié par la méthode directe, puis par la méthode adaptée. On a

observé un gain en mémoire de l'ordre du nombre d'itérations des boucles indépendantes, et cela pour un coût minime en temps d'exécution.

Ces résultats ont été présentés à la conférence AD2000, et seront publiés dans les actes de cette conférence.

Au centre de cette étude se trouve une propriété essentielle des programmes adjoints obtenus par différentiation automatique: il existe un isomorphisme bien précis entre le graphe de dépendance des données d'un programme et celui de son programme adjoint. Nous sommes en train de développer une preuve formelle de cet isomorphisme.

6.2 Extension d'IL aux constructions objet

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

Participants : Martial Bonaventure, Valérie Pascual, Frédéric Olier, Laurent Hascoët.

Pour construire la plate-forme d'analyse de langages impératifs ALI, on a défini 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++. Jusqu'à maintenant, les constructions orientées objet n'étaient pas prises en compte. Dans le cadre du stage de Martial Bonaventure, nous avons étudié les extensions à apporter à IL et à ALI pour traiter correctement l'essentiel des constructions objet.

Cela a demandé une modification du langage abstrait IL, pour représenter la définition des classes, de leurs champs et méthodes, et aussi les appels de méthodes. D'autre part les représentations internes des programmes ont été perfectionnées en conséquence. L'une des extensions essentielles concerne la table des symboles. Nous avons proposé une architecture de la table des symboles qui capture les notions d'héritage simple et multiple entre les classes, ainsi que les différentes visibilités (privé, public, etc...).

6.3 Analyse Lecture-Ecriture dans ALI

Mots clés : analyse statique, lecture-écriture.

Participants : Jean-Philippe Sautarel, Valérie Pascual, Frédéric Olier, Laurent Hascoët.

L'analyse « lecture-écriture » est une analyse statique très classique. Elle consiste, pour un fragment de programme donné, à classer les variables de ce fragment en quatre catégories:

- les variables dont la valeur en entrée du fragment est seulement utilisée dans le fragment,
- les variables dont la valeur en entrée du fragment n'est pas utilisée, mais est redéfinie dans le fragment,
- les variables dont la valeur en entrée est utilisée puis redéfinie,
- les variables qui ne sont ni utilisées, ni redéfinies dans le fragment.

Cette analyse a un intérêt général, et devra être effectuée systématiquement par ALI. Par exemple en Différentiation Automatique, elle est utile pour le *checkpointing*. Cette technique

implémente un compromis entre stockage et recalcul, et par conséquent peut conduire à exécuter une même phase du calcul plusieurs fois. Pour cela, il est nécessaire de mémoriser toutes les valeurs nécessaires à cette phase. L'analyse « lecture-écriture » permettra de déterminer statiquement un ensemble réduit de variables dont la valeur doit être stockée.

Dans le cadre du stage de Jean-Philippe Sautarel, nous avons implémenté cette analyse dans ALI. Nous avons utilisé pour cela la machinerie interprocédurale d'ALI. Il s'agit de la troisième utilisation différente de cette machinerie, ce qui nous a conduit à clarifier son API.

Comme toutes les analyses statiques, l'analyse « lecture-écriture » rencontre ses difficultés principales lorsque les variables sont des tableaux et lorsque le contrôle dynamique (conditionnelles, boucles...) n'est pas connu statiquement. Dans ce cas, on a très vite des informations partielles (variable *peut-être* lue ou écrite...). La combinaison de ces informations devient délicate. De plus, ces informations sont calculées itérativement sur le programme, jusqu'à atteindre un point fixe, ce qui pose le problème de la terminaison. Nous avons proposé un raffinement de la représentation de ces informations pour lequel on peut prouver que les informations calculées sont croissantes au cours des itérations, à l'intérieur d'un ensemble fini, et donc qu'un point fixe est nécessairement atteint en un temps fini.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 EDF: performances d'Odyssée

Participants : Frédéric Olier, Valérie Pascual, Laurent Hascoët.

L'année précédente, dans le cadre du contrat avec EDF, nous avons identifié le problème de l'inefficacité d'ODYSSÉE sur les gros programmes. Cela provenait de la phase de détection des dépendances numériques, dont la complexité croissait plus vite que linéairement par rapport au nombre de variables présentes. D'une part les variables étaient représentées par leur nom, et d'autre part les informations étaient stockées sous la forme de listes d'associations indexées par ces noms. Ces choix conduisent à des implémentation notoirement inefficaces. De plus, représenter les variables par leur nom pose de gros problèmes pour le traitement des équivalences.

Plutôt que de modifier ces choix fondamentaux à l'intérieur d'ODYSSÉE, nous avons préféré supprimer cette analyse dans ODYSSÉE, et la sous-traiter à ALI. La représentation des variables dans ALI leur associe un indice unique, ce qui résout les questions d'équivalences. Les informations attachées aux variables deviennent des tableaux indexés par cet indice, et l'accès à ces informations est alors extrêmement rapide. Enfin, ces informations sont très souvent codées par quelques booléens, et les opérations sur des tableaux de booléens sont implémentables d'une manière extrêmement efficace.

Une nouvelle version d'ODYSSÉE interfacée avec ALI a été réalisée et livrée à EDF. Nous l'avons testée sur les plus gros programmes à notre disposition. La détection des dépendances numériques est dorénavant linéaire par rapport au nombre de variable. Pour de gros programmes (THYC3D), cela fait diminuer le temps de différentiation, de plusieurs heures à une vingtaine de minutes.

Dorénavant, avec cette nouvelle architecture, la détection des dépendances numériques

traite correctement les `EQUIVALENCE`, les `COMMON` dont les déclarations diffèrent d'une routine à l'autre, les variables rémanentes, ainsi que les types de données structurées. Le passage de ces informations aux frontières entre les sous-programmes a demandé la mise en place de deux formalismes de représentation, l'un interne à une routine donnée, l'autre externe, indépendant de la routine appelante, ainisi que les procédures mécaniques de traduction d'une forme à l'autre.

L'interfaçage entre ODYSSÉE et ALI a constitué un premier test réussi de l'API de la plate-forme ALI.

7.2 EDF: Langage de commande

Participants : Frédéric Olier, Laurent Hascoët.

Les contrats d'étude précédents entre l'INRIA et EDF portaient sur la différentiation de codes exclusivement écrits en FORTRAN. Or, de nombreux codes chez EDF consistent en un ensemble de fragments de code FORTRAN, reliés entre eux par un langage de commande, qui assure le séquencement des fragments et le passage des données entre eux. Le but de cette étude est d'étudier la différentiation automatique de ces programmes composites.

Dans le cas particulier proposé par EDF, le langage de commande est PYTHON, qui appelle une couche logicielle nommée SUNSET, définie en C et FORTRAN, qui appelle enfin des procédures numériques écrites en FORTRAN.

Nous avons identifié deux problèmes majeurs:

- l'exécution finale de l'application est guidée, non pas par un code explicite en PYTHON, C ou FORTRAN, mais par des textes de commandes, qui sont des chaînes de caractères, dynamiquement analysées et exécutées par SUNSET,

- les données ne sont pas rangées dans un ensemble défini de variables et de tableaux, mais dans une structure associative dynamique. Les données sont repérées dans cette structure au moyen de noms qui se trouvent dans les textes de commande.

Pour différentier ces applications, il nous a semblé plus sage, non pas de différentier directement cette application multi-langage dont le contrôle est mal connu statiquement, mais d'effectuer une phase préliminaire d'évaluation partielle.

La conclusion de ce travail est un rapport préconisant une évaluation partielle préalable. Les données seraient les textes de commande, mais pas les données numériques. Le résultat de cette évaluation partielle est donc un programme spécialisé pour un ensemble fixé, donné de textes de commandes, et qui effectue le calcul numérique lorsqu'on lui fournit les données numériques manquantes. Ce programme est donc différentiable, et conduit aux mêmes dérivées qu'une différentiation simple.

Les calculs numériques proprement dits étant tous effectués en FORTRAN, le résultat de l'évaluation partielle est un programme mono-langage.

Enfin, la réalisation de cet outil d'évaluation partielle demanderait un effort de plusieurs années. Pour en démontrer le principe, nous avons réalisé une petite maquette qui fonctionne sur un petit nombre d'exemples.

7.3 Participation à des colloques, séminaires, invitations

- présentation des travaux du projet à un séminaire du projet A3 le 13 janvier 2000 (Laurent Hascoët).
- participation à la réunion DASSAULT-INRIA organisée à Sophia-Antipolis le 20 janvier 2000 (Laurent Hascoët).
- invitation à venir présenter les travaux du projet devant l'équipe d'Andreas Griewank à l'université de Dresde du 22 au 25 mars (Laurent Hascoët).
- présentations au workshop de Différentiation Automatique organisé à l'université du Hertfordshire/Hatfield (GB) le 3 mai 2000 (Christèle Faure et Laurent Hascoët).

8 Diffusion de résultats

8.1 Animation de la communauté scientifique

Le projet a organisé la 3^{e} conférence internationale sur la Différentiation Automatique, qui s'est tenue à Nice du 19 au 23 juin 2000. `http://www-sop.inria.fr/tropics/ad2000/`. Cette conférence a réuni l'ensemble des équipes travaillant sur la Différentiation Automatique à travers le monde. Il y a eu plus de 60 présentations, plusieurs «panel discussions» et réunions de groupes d'intérêt, qui ont réuni un total de 101 participants. Les actes sont en cours d'édition, et seront probablement publiés dans la série LNCSE, chez Springer.

9 Bibliographie

Ouvrages et articles de référence de l'équipe

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

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

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

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

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

[6] L. HASCOËT, «Parallelization of Finite Element Codes with Automatic Placement of Communications», *rapport de recherche n^{o} 3646*, Inria, 1999.

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

Communications à des congrès, colloques, etc.

[8] B. CAPPELAERE, D. ELIZONDO, C. FAURE, «Odyssée- versus Hand- Differentiation of a Terrain-Modelling Application», *in : Proceedings of AD2000, Nice, France*, 2000.

[9] C. FAURE, U. NAUMANN, «Minimizing the Tape Size», *in : Proceedings of AD2000, Nice, France*, 2000.

[10] L. HASCOËT, S. FIDANOVA, C. HELD, «Iteration-wise Adjoining», *in : Proceedings of AD2000, Nice, France*, 2000.

[11] E. SOULIÉ, C. FAURE, T. BERCLAZ, « Electron Paramagnetic Resonance, Optimization and Automatic Differentiation», *in : Proceedings of AD2000, Nice, France*, 2000.

Rapports de recherche et publications internes

[12] C. FAURE, E. SOULIÉ, T. BERCLAZ, «Résonance paramagnétique électronique, optimisation et différentiation automatique», *Rapport de recherche n° 3907*, INRIA, 2000, `http://www.inria.fr/rrrt/rr-3907.html`.