Proposition de Projet
TROPICS
Transformations et Outils Informatiques pour le Calcul Scientifique

10 juin 1999

Abstract

Nous proposons un projet, nommé TROPICS, dont l'objet est d'étudier et concevoir des outils logiciels spécifiquement adaptés aux problèmes rencontrés par les concepteurs et développeurs d'applications de calcul scientifique. Ces applications de calcul scientifique imposent des contraintes importantes, provenant de leur taille d'une part, des langages et des styles de programmation employés d'autre part. Par ailleurs, le domaine du calcul scientifique recèle des besoins en analyses et transformations de code, beaucoup plus variées et complexes que les classiques vérifications de typage et traductions. Nous proposons donc de tirer parti de deux aspects de notre expérience, qui sont l'analyse, la compilation et la transformation semi-automatiques de programmes d'une part, la compréhension des problèmes du calcul scientifique d'autre part, pour proposer, étudier et implémenter des outils logiciels nouveaux, utiles aux acteurs du calcul scientifique.

Contents

1 Composition de l'équipe

- **Chercheurs permanents:**
  - **Laurent Hascoët** (CR INRIA, responsable scientifique)
  - **Valérie Pascual** (CR INRIA)
- **Chercheur contractuel:**
  - **Christèle Faure** (CR INRIA "contractuel" *($\Rightarrow$ 8/00)*)
- **Professeur associé:**
  - **Andreas Griewank** (Visiteur pour un an *($\Rightarrow$ 8/99)*)
- **Etudiants post doctoraux:**
  - **Patrick Dutto** (Post-Doc GÉNIE *($\Rightarrow$ 1/00)*)
  - **Stefka Fidanova** (Post-Doc DÉCISION *($\Rightarrow$ 2/00)*)
  - **Uwe Naumann** (Post-Doc MIO *($\Rightarrow$ 1/00)*)
  - **Frédéric Olier** (Post-Doc, contrat EDF *($\Rightarrow$ 5/00)*)

2 Introduction

2.1 Description du projet

Nous proposons de rassembler une équipe de personnes ayant une expérience approfondie des questions liées aux *analyses* et *transformations semi-automatiques* des programmes. Cette équipe aura également une bonne compréhension des problèmes spécifiques du *calcul scientifique*. Notre but est de développer et prolonger les recherches qui concernent les outils d'aide aux transformations de programmes scientifiques.

Dans cette proposition, nous commençons par décrire le domaine scientifique du projet (2.1.1), ainsi que sa motivation principale, qui est de répondre aux besoins logiciels ressentis par les acteurs du calcul scientifique (2.1.2). Nous décrivons alors les objectifs du projet (2.1.3), avec une priorité accordée à la différentiation automatique et à la parallélisation, suivis d'autres directions plus prospectives, la comparaison et la synthèse.

La section 2.2 décrit l'état de l'art dans les domaines en rapport avec notre proposition. À nouveau l'accent est mis sur les techniques généralistes pour les transformations de programme (2.2.1), et sur les questions specifiques à la différentiation automatique (2.2.2) et la parallélisation (2.2.3).

Dans la section 3, nous présentons les questions de recherche, en commençant par présenter en quoi l'expérience scientifique des membres du projet (3.1) les rend aptes à adresser les problèmes visés par le projet. Puis nous présentons le programme de recherche (3.2).

La section 4 présente les questions liées au développement, en commençant par présenter les outils et programmes existants (4.1). Puis nous présentons les développements nouveaux probables (4.2), ainsi qu'une proposition d'architecture pour faire coopérer les deux outils principaux ODYSSÉE et PARTITA.

La section 5 donne une ébauche de répartition du travail, chronologiquement, et par rapport aux membres su projet.

La section 6 présente les liens du projet avec l'extérieur, en commençant par les rapports avec les autres projets de l'INRIA, suivie par les liens académiques que nous désirons nouer ou renforcer, notamment pour inspirer nos recherches, en amont, et diffuser nos résultats, en aval. Pour terminer, nous présentons les liens industriels, qui favorisent la valorisation de nos travaux.

2.1.1 Domaine scientifique

Le domaine scientifique de ce projet est donc celui des *manipulations semi-automatiques* de programmes. Ici, on conçoit des algorithmes et des programmes dont les objets sont eux-mêmes des programmes. En particulier, ce domaine couvre les questions liées à la *compilation*. Les programmes étant donnés sous leur forme textuelle, compréhensible par l'utilisateur, ils doivent tout d'abord être traduits dans une forme interne d'*arbre syntaxique*, au moyen d'un *analyseur syntaxique*, ou *"parseur"*. Il est souvent nécessaire de recourir à des formes encore plus abstraites, telles que des *graphes*, par exemple de *flot de contrôle*. On implémente alors les analyses en utilisant des techniques proches de l'*interprétation abstraite*.

Une caractéristique essentielle de ce projet est que nous nous concentrerons sur les transformations utiles aux programmes scientifiques. Cela nous obligera à considérer des programmes de grande taille, composés de nombreux sous-programmes. De plus, il sera obligatoire de traiter correctement l'ensemble des caractéristiques des langages traités (boucles, pointeurs...) et de ne pas se limiter à des sous-langages simplifiés. En revanche, on tirera profit des observations suivantes:

- Tous ces programmes sont écrits dans des *langages impératifs*, tels que FORTRAN ou C, dans lesquels le contrôle et les structures de données sont décrits *explicitement*.
- Les bases théoriques mathématiques sont très développées et complexes, et il existe un lien conceptuel très étroit entre la théorie et l'implémentation.
- L'utilisation de bibliothèques de calcul scientifique spécialisées est très répandue.

2.1.2 Motivation: les besoins du calcul scientifique

Le Calcul Scientifique est un important créateur de programmes informatiques, et un grand consommateur de capacités de calcul. Mais les acteurs de ce domaine, de par leur culture scientifique, s'intéressent plus aux problèmes théoriques et mathématiques qu'aux problèmes informatiques qu'ils rencontrent. On remarque d'autre part que ces problèmes informatiques sont souvent communs aux divers domaines du calcul scientifique.

La qualité informatique des programmes de calcul scientifique est en général très bonne (normes de programmation, commentaires...). Cependant, ces programmes sont de grande taille, et doivent évoluer en permanence. Ils doivent suivre l'évolution des théories, résoudre des problèmes physiques plus complexes, par des manières plus performantes telles que maillages adaptatifs, méthodes multigrilles, ou calculs de gradients. Ils doivent aussi suivre l'évolution des machines (parallélisme...). Tout cela mène à des programmes tellement complexes qu'il n'est plus réaliste d'effectuer toutes ces transformations manuellement. L'accord est général pour reconnaître un besoin important d'outils d'aide à la synthèse, la maintenance ou la transformation de programmes.

2.1.3 Objectifs du projet

Plusieurs outils pour le calcul scientifique sont déjà identifiés. Leur étude a déjà été entamée, plus ou moins loin, dans le cadre d'autres projets (*cf* sections 3.1 et 4.1). Ils seront naturellement parmi les premières cibles du nouveau projet.

À court terme, le projet s'intéressera principalement à deux classes d'outils, autour de:

- la Différentiation Automatique (D.A.), et notamment sur la mise en valeur des points forts du logiciel **Odyssée** [90] [42], développé par le projet SAFIR et l'Université de Nice depuis 1992. Il faudra étudier les points suivants: consommation mémoire, stratégies de stockage/recalcul, synthèse du mode inverse sur le graphe de flot, vectorisation et matrices jacobiennes creuses.

- la Parallélisation (Semi-)Automatique, et notamment les méthodes de parallélisation en faveur chez les numériciens, comme la parallélisation SPMD *(Single Program Multiple Data)* par partition de maillage. Le lien avec la D.A. sera aussi étudié, notamment pour la différentiation autour des instructions de communication et synchronisation.

Pour ces outils, le projet devra également maintenir une compétence dans des techniques de base, telles que la détection et la propagation de propriétés des programmes, la simplification, ou les problèmes de manipulations interprocédurales. Pour cela, le projet dialoguera avec d'autres projets spécialistes par exemple d'évaluation partielle et slicing (*cf* section 2.2.4).

À plus long terme, deux autres classes d'outils nous paraissent intéressantes. Il s'agit de:

- la comparaison et la reconnaissance de routines, c'est-à-dire reconnaître automatiquement que deux fragments de programme ont la même sémantique.

- la synthèse de portions de programmes à partir de spécifications de plus haut niveau (*e.g.* d'équations)

Nous nous plaçons ici dans une approche plus prospective. Ces sujets ont été assez peu étudiés à notre connaissance, et un travail de recherche important est nécessaire pour défricher ces domaines.

Notons bien que les outils énumérés ici ne forment pas une liste hétérogène d'outils indépendants. Bien au contraire, il y a de profondes relations, que nous soulignons:

- Les analyses préliminaires sont communes.

- La dimension interprocédurale est essentielle.

- La représentation interne des programmes est commune, et probablement assez indépendante du langage choisi.
- L'utilisation des outils est souvent couplée, les uns utilisant les résultats des autres, parfois avec des cycles.

Par conséquent, un but du projet, à moyen terme, est de définir un environnement commun pour la description et l'implémentation de transformations variées. Il faudra en particulier *enrichir la représentation interne* des programmes impératifs pour permettre une bonne description des transformations. En particulier, les arbres syntaxiques sont rarement la forme abstraite la plus adaptée aux analyses et transformations, et nous devrons utiliser ou inventer des formes dérivées des graphes de flot de contrôle (FG). De plus il faudra *définir un mode de communication* des informations sur les programmes, par le moyen d'une *interface* formelle, pour permettre le couplage des diverses transformations, entre elles ou dans un environnement externe. Cet environnement permettra alors de développer de nouveaux outils. Notre inspiration sera alimentée notamment en détectant les besoins logiciels utiles au calcul scientifique, ou à certains sous-domaines, en collaboration étroite avec les spécialistes de ces domaines. Puis une étude généralisera chaque besoin, le formalisera en lui trouvant un cadre théorique. On spécifiera alors un outil, en cherchant des algorithmes et/ou des interfaces utilisateur. Au vu des caractéristiques des programmes scientifiques, il est prévisible que les questions d'*efficacité* seront d'une grande importance. Cela vaut tant pour l'outil que pour ses résultats (programmes générés). Enfin, l'environnement permettra d'implémenter des prototypes mettant en évidence les points forts des algorithmes. Le retour d'utilisation par les acteurs du calcul scientifique sera particulièrement recherché.

2.2 État de l'art

2.2.1 Analyses et transformations en général

L'analyse et la transformation de programmes sont des activités anciennes et classiques. Ce sont entre autres les opérations essentielles des compilateurs [4]. 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 [87] et CENTAUR [18] [1]. 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 utilisée dans les compilateurs 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, et autorise principalement

- des traductions non optimisées, sans analyse globale,
- des analyses limitées aux programmes parfaitement structurés,
- des interprètes limités aux programmes structurés, sauf acrobaties pour traiter des `GOTO` ou même des `letrec` de ML.

Concrètement, il y a trop d'analyses fondées sur des graphes pour se contenter d'une représentation d'arbres. Reconnaissons qu'en contrepartie, les analyses et transformations sur les arbres [68] [9], 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 [31], ou pour des optimisations globales (code mort, variables vivantes, code invariant...).

La plupart des outils réels pour programmes impératifs, utilisent des analyses reliant les différents usages d'une même variable. Cela est utile autant pour la parallélisation, que pour le slicing (*cf* section 2.2.4) ou la différentiation. La forme classique pour représenter ces relations est le graphe de dépendances. Il en existe plusieurs variantes, dont le *data-flow graph*, le *data-dependence graph* [71] [5], le *program dependence graph* [45] ou le *dependence flow graph* [84]. L'utilisation de ces graphes dans le cas particulier de la parallélisation est décrit dans [95], [43], [5] ou [33], et pour la comparaison de sous-programmes dans [86] ou [6].

Considérant l'exemple CENTAUR, on peut déplorer le trop petit nombre d'applications en vraie grandeur, c'est-à-dire produisant des outils s'appliquant à des programmes réels, de manière efficace. On soulignera cependant les exceptions notables que sont les outils dérivés autour de COQ [12], JAVA [8] et FORTRAN [48] [3]. Pour notre projet, l'outil FORESYS constituera une référence. Il montre en particulier l'intérêt des graphes de flot dans les analyses. De plus, un effort considérable a dû être fourni pour mettre en place les analyses interprocédurales de FORESYS. Cela met en évidence à quel point un mécanisme générique de gestion des analyses interprocédurales, est nécessaire, et fait défaut dans CENTAUR.

Les outils de manipulation de programmes se conçoivent naturellement comme un enchaînement d'analyses et de transformations. Souvent, des outils différents partagent certaines opérations. Cela doit entraîner des choix d'architecture. Les recherches autour de CENTAUR ont étudié cette question en détail [28], et montré l'intérêt d'une conception *distribuée* des outils de transformation.

Pour continuer cette discussion sur l'état de l'art, nous devons maintenant examiner, au cas par cas, les outils énumérés dans la section 2.1.3. Cela ne signifie pas pour autant que ces outils soient indépendants. Il s'agit simplement du reflet de la situation actuelle, où ces outils sont étudiés séparément par des groupes qui communiquent peu. Nous aimerions que notre projet puisse contribuer à corriger cette situation.

2.2.2 Différentiation Automatique

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. L'étude de la D.A. a une longue tradition à l'INRIA [52]. Il existe à l'heure actuelle deux approches pour obtenir P'. Soit par une transformation de source à source (outils ODYSSÉE [42], ADIFOR [14] [13], TAMC [49] [50], PADRE2 [2] [70]), soit par surcharge des opérateurs arithmétiques (outils ADOL-C [56], ADOGEN [89]), dans laquelle les opérateurs =, *, *etc* sont modifiés pour effectuer en même temps les opérations sur les valeurs et leurs dérivées. Les dérivées calculées sont ensuite utiles dans plusieurs types d'applications, comme

a) les études de sensibilité [21] [37] [40],

b) l'analyse des propagations d'erreurs d'arrondi [67] [66] [70].

c) les problèmes inverses tels que l'assimilation des données [72] [94] [91],

d) les méthodes d'optimisation de formes [74], les méthodes d'optimisation sous contraintes, et plus généralement tous les algorithmes basés sur une linéarisation locale.

e) la mise en œuvre de l'itération de Newton [53],

f) l'intégration implicite de problèmes d'évolution (équations différentielles ordinaires ou équations aux dérivées partielles), en particulier pour les problèmes raides [23].

Aux utilisations variées des programmes dérivés, répondent divers *modes de différentiation*. L'état de l'art varie selon ces modes. Dans ce qui suit, nous appelons P un programme initial, $e_i, i \in [1, n]$ 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 $\mathbb{R}^n$. Cela revient également à considérer la fonction linéaire "tangente" à f au point $\{e_1, ..., e_n\}$, et à l'appliquer au vecteur $\{de_1, ..., de_n\}$. La plupart des outils existants de D.A. proposent ce mode "fondamental". Il a été relativement bien étudié d'un point de vue théorique. On a une bonne estimation de sa complexité et de sa consommation mémoire [66]. Divers raffinements ont été proposés, tels que le calcul simultané pour plusieurs directions de variation des entrées. Ce mode est implanté dans la plupart des outils de Différentiation Automatique.

- On peut également imaginer un *mode "matrice jacobienne"*, qui construirait 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 jacobienne est utile aux applications **d)** et **e)**. Ce mode n'est pas fourni par les outils actuels, sauf par ADIFOR [14]. Une expérience a été realisée dans ODYSSÉE. 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'. En réponse, on peut exploiter le fait que certaines jacobiennes sont creuses, et les manipuler en conséquence. On peut aussi calculer des dérivées directionnelles (*mode "direct"*) suivant certaines directions bien choisies, puis reconstruire la matrice [30] [73] [32] [85]. ADIFOR exploite cette propriété au moment de l'exécution, en sous-traitant systématiquement les opérations sur les jacobiennes à une bibliothèque spécialisée dans les matrices creuses. On peut aussi tenter de trouver la structure creuse de la jacobienne par une exécution "observée" de P, ce qui est à rapprocher des méthodes "inspector-executor" [69], [20], utilisées en parallélisation. On peut enfin tenter de trouver cette structure creuse statiquement, si P est suffisamment régulier, par des méthodes inspirées des analyses de parallélisation [93]. Cette structure creuse peut ensuite être exploitée de diverses manières [32], [81] dans P'. D'ailleurs, la jacobienne peut aussi être calculée en utilisant le mode précédent (direct), en fournissant comme ensemble de directions de différentiation, la base canonique de $\mathbb{R}^n$.

- Le *mode "Taylor"* construit le programme

$$P' : \{e_1, ..., e_n, de_1, ..., de_n\} \longmapsto \{r_1, ..., r_m, d^k r_1, ..., d^k r_m\}$$

qui calcule les coefficients, jusqu'à un certain degré, de la formule de Taylor exprimant $f(\{e_i + de_i\}) - f(\{e_i\})$. Cette formule fait intervenir les tenseurs des dérivées d'ordre supérieur de f. Ce mode est utile pour les applications **d)** et **h)**. Il est proposé dans les outils ADOGEN [89] et ADOL-C [56]. Il fait l'objet de recherches pour évaluer et améliorer sa complexité. Par exemple, l'équipe d'Andreas Griewank à Dresde, a montré comment remplacer le calcul explicite des tenseurs dérivés, par un calcul de plusieurs formules de Taylor à une seule variable, suivi d'une interpolation [58]. Ceci exploiterait naturellement les symétries des tenseurs dérivés d'ordre 2 et plus.

- Le *mode "inverse"* construit le programme

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

  qui calcule, pour une certaine pondération des résultats dr_j^*, la direction de variation de_i^* des entrées, qui maximise la variation des résultats pondérés. Ceci revient à dire que l'on calcule la direction de sensibilité maximale de f, ou encore son gradient. Ce mode est actuellement proposé par les outils ADOL-C [56], TAMC [49] et ODYSSÉE [42], PADRE2 [2]. Il est utilisé pour les problèmes **c)** et **d)**. Ce mode présente un avantage dans le cas où la routine P a un petit nombre de résultats ($m << n$). Plus précisément, la complexité du mode direct est liée au nombre d'entrées n, alors que celle du mode inverse est liée au nombre de résultats m [66]. Cependant, le mode inverse utilise les valeurs intermédiaires calculées par P, dans l'ordre *inverse*. 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 [26]. Les compromis stockage/recalcul font aussi l'objet de recherches actives [54], [59], [25].

Par ailleurs, il y a des recherches sur la combinaison des modes direct et inverse [57]. Dans le cas particulier de la différentiation par surcharge, l'exécution du programme P génère une "trace", qui représente l'enchaînement des opérations réellement effectuées, et qui doivent être différentiées. C'est le graphe de calcul ("computational graph"). Le calcul des dérivées sur ce graphe peut être fait dans des ordres différents, correspondant aux modes direct, inverse, ou à une combinaison quelconque. Trouver des ordres particulièrement efficaces fait l'objet de recherches pratiques et théoriques [76] [15].

Le lien avec la parallélisation est également étudié, tant pour une différentiation de programmes parallèles, qui conserverait le parallélisme existant, que pour générer des programmes différentiés qui exploitent une forme de parallélisme.

Il y a aussi des recherches sur les cas limites de la différentiabilité des programmes. Le programme P représentant en fait une fonction différentiable par morceaux, on étudie ce qui se passe aux frontières entre les morceaux. Dans certains cas, une dérivée directionnelle peut alors être valide.

L'étude des interactions avec l'utilisateur est encore balbutiante. Le besoin existe cependant, par exemple, de laisser l'utilisateur spécifier les dérivées qu'il désire ou ne désire pas, les opérations ou parties de programmes dont il fournira la dérivée lui-même, ou le choix des points de stockage/recalcul pour le mode inverse. Plus généralement, on s'accorde à croire qu'un outil opérationnel de D.A. ne fonctionnera que s'il est capable d'exploiter des informations fournies par l'utilisateur, mais on ne sait pas bien énumérer ces informations, ni la façon la plus pertinente de les demander à l'utilisateur.

2.2.3 Parallélisation

Le domaine de la parallélisation est très vaste et foisonnant. Il est aussi en évolution constante, rapide, suivant l'architecture des machines. Il importe de ne pas confondre, d'une part la *programmation parallèle*, qui consiste à écrire une application en appliquant un style de programmation parallèle donné, et d'autre part la *parallélisation*, qui consiste à transformer un programme existant pour l'adapter à un tel style parallèle, en particulier en utilisant un outil. Nous nous intéressons ici aux outils de parallélisation. Il existe de très nombreux styles de programmation parallèle, et nous ne prétendons pas être exhaustifs.

Il existe un parallélisme à "grain fin", dans lequel on peut désigner des (parties d')instructions, dont l'ordre relatif d'exécution n'a pas d'importance. Cela recouvre en particulier la vectorisation, le `FORALL` de HPF et de FORTRAN95, ou les boucles explicitement parallèles, comportant éventuellement des barrières de synchronisation. On s'intéresse aux itérations d'une même instruction dans une boucle. Classiquement, on détecte ces (parties de) boucles parallélisables par une analyse des *dépendances* entre les opérations [95]. Les *cycles* dans le graphe des dépendances indiquent les instructions non parallélisables. Plusieurs outils commerciaux, tels que VAST [83], FORGE [7], ou PARTITA [63], proposent ce type de parallélisation.

Il existe aussi un parallélisme à "gros grain", où on désigne de grosses parties du programme ("tâches") qui peuvent s'exécuter indépendamment. Similairement, chaque tâche peut être un même programme, mais s'exécutant pour chaque tâche sur des données différentes (parallélisme SPMD). La parallélisation correspondante utilise également le graphe de dépendances, mais cette fois, une dépendance entre deux tâches se traduit par l'insertion d'une communication, qui synchronise les deux tâches, et qui ralentit le programme. Cette détection des communications peut être faite statiquement pour des programmes réguliers, ou dynamiquement pour des programmes irréguliers. Dans ce dernier cas, une première exécution "à vide" détecte et place les communications (méthode "inspector-executor" [69]).

Mais le problème principal actuellement est moins la parallélisation des opérations que la localité des accès mémoire. En effet, les architectures des machines ont évolué vers une mémoire distribuée entre les processeurs. La taille des problèmes à traiter rendait cette évolution nécessaire. Cela conduit à de nouveaux problèmes: même si une boucle est parallèle, l'exécution sur un processeur peut demander des accès aux mémoires des autres processeurs. Or le temps d'un accès mémoire est maintenant prépondérant par rapport aux opérations de calcul. Cela peut rendre l'exécution parallèle *encore plus lente* que l'exécution séquentielle! Il faut donc trouver un placement des données sur les processeurs qui limite les communications. En HPF par exemple, des directives permettent à l'utilisateur de spécifier cet alignement. Mais peut-on *déduire* un bon alignement en analysant le programme lui-même? Il s'agit d'un problème difficile, qui n'a que des solutions approchées. À nouveau, on utilise le graphe des dépendances. De tels outils expérimentaux sont décrits par exemple dans [10]. À la limite, le problème se pose en l'absence de parallélisme. Sur une machine séquentielle disposant d'un cache, on peut améliorer énormément le temps d'exécution en transformant le programme, en regroupant les accès mémoire pour limiter les défauts de cache.

Dans tous les cas, ces transformations nécessitent une bonne analyse des dépendances, principalement entre les références aux tableaux. Lorsque les indices des tableaux sont des fonctions simples (e.g. linéaires) des compteurs des boucles, on dispose d'outils puissants pour analyser les dépendances [44], [92]. On peut alors connaître le nombre d'itérations qui séparent les deux opérations dépendantes, ce qui est une information essentielle pour des parallélisations raffinées, telles que changement de coordonnées [19] ou tiling.

Cependant, les programmes réels ayant souvent des indices de tableaux non linéaires, on ne peut pas se contenter de ces méthodes. Dans les outils de parallélisation industriels, la réponse est souvent de laisser l'utilisateur donner des directives à l'outil.

Pour ce qui concerne l'aspect interprocédural, les outils qui en tiennent compte se limitent le plus souvent à exploiter une signature simple des procédures appelées. Cette signature indique par exemple, pour chaque variable, si elle est lue et/ou réécrite dans la procédure.

2.2.4 Évaluation partielle et slicing

L'évaluation partielle consiste à propager dans un programme des informations (partielles) sur ses données, et cela au cours d'une analyse statique. De cette manière, on espère trouver des simplifications du programme, telles que des simplifications d'expressions arithmétiques, ou des tests toujours vrais ou faux. Le résultat de l'évaluation partielle est donc un programme qui a la même sémantique que le programme initial, sur un sous-ensemble de ses données possibles. Ce sujet est déjà attaqué par d'autres projets de l'INRIA comme COMPOSE.

Réciproquement, le slicing consiste à propager dans un programme l'information selon laquelle certaines variables résultat sont inutilisées. Cette propagation se fait donc à rebours. De cette manière, on espère trouver des simplifications telles que l'élimination d'instructions qui ne calculent que des variables inutilisées. Le résultat du slicing est donc un programme qui donne les mêmes résultats que le programme initial, mais seulement pour un sous-ensemble donné des résultats.

De notre point de vue, l'évaluation partielle et le slicing sont deux opérations duales et peuvent être étudiées ensemble. Ces transformations peuvent avoir plusieurs usages. Une utilisation ambitieuse est de générer automatiquement des compilateurs à partir d'interprètes [47]. Une utilisation plus concrète est d'aider à la compréhension de programmes. En effet, les gros programmes combinent souvent en un seul code plusieurs options relativement disjointes. Lorsque de plus ces programmes sont anciens, il peut être très difficile de démêler ces programmes. L'évaluation partielle et le slicing sont alors utiles [16]. Nous voyons aussi un lien avec la différentiation automatique, pour ce qui concerne la propagation des variables actives et des résultats à dériver.

Ces transformations doivent s'attaquer à deux problèmes principaux, que nous rencontrerons aussi pour la plupart des outils. D'une part, il faut propager les informations à travers un sous-programme donné, de la manière la plus fine possible. L'outil le mieux adapté pour cela est l'interprétation abstraite sur le graphe de flot de contrôle. Alors que cette propagation est relativement aisée pour le slicing, elle est délicate pour l'évaluation partielle. En effet il faut calculer intelligemment de quelle manière une contrainte sur les variables d'une expression se traduit en une contrainte sur son résultat. À cause des conditionnelles et des indices de tableaux complexes, il est presque toujours nécessaire d'effectuer des *unions* de contraintes, ce qui rend, bien entendu, une contrainte plus faible. Il faut également s'assurer que la propagation termine, en montrant qu'il n'existe pas de chaine infinie de contraintes strictement de plus en plus faibles.

La deuxième classe de problèmes concerne la propagation interprocédurale. La propagation se faisant depuis la routine principale, elle est nécessairement descendante dans le graphe d'appel. Que doit-on faire quand un même sous-programme S est appelé plusieurs fois? Doit-on le spécialiser, et générer autant de transformations de S que de cas d'appel, ou bien doit-on généraliser tous les cas d'appel en un seul, moins précis, ce qui ne génèrera qu'un seul sous-programme simplifié pour S ? Bien entendu, tout cela est encore plus ardu pour les appels situés dans des boucles, ou pour les programmes récursifs. On peut laisser le choix à l'utilisateur par une interface ou utiliser des heuristiques [17].

2.2.5 Comparaison de programmes

Nous considérons le problème de comparer deux (fragments de) programmes donnés. Le but est d'évaluer à quel point ils sont proches du point de vue sémantique. Plus généralement, nous considérons le problème de trouver dans un programme donné, les fragments proches d'un fragment donné. À notre connaissance, ce problème – non décidable en général – a été relativement peu étudié.

Frilley [46] propose une comparaison d'arbres syntaxiques, qui a l'avantage de faire le lien avec des problèmes classiques de reconnaissance de chaînes de caractères. Cependant, aucune notion sémantique n'est prise en compte. Reps [86] propose une comparaison tenant compte de la sémantique, d'une manière limitée. L'utilisation visée est une comparaison intelligente des versions successives d'un programme. Cette comparaison ne recherche que l'équivalence stricte, et non un degré de ressemblance. De plus le critère d'équivalence choisi est encore trop fort (il considère différents certains programmes pourtant sémantiquement équivalents).

Dans sa thèse, Laurent Angeli [6] propose des algorithmes pour comparer sémantiquement deux fragments de programme. Cette comparaison donne le degré de ressemblance des deux fragments, ce qui permet par exemple d'extraire un sous-programme unique réunissant les sémantiques des deux fragments. Si la ressemblance est totale (aucune différence), ces algorithmes peuvent alors reconnaître des implémentations d'opérations simples, telles que produits de matrices, ce qui peut aider à la compréhension et à l'optimisation des programmes.

2.2.6 Synthèse à partir de spécifications

Pour l'essentiel, les tentatives de synthèse automatique à partir d'équations, ont été faites à l'intérieur d'outils de calcul formel ou de calcul scientifique, comme MAPLE ou SCILAB. De plus il s'agit de générer des sous-programmes relativement courts, qui implémentent l'évaluation d'une expression mathématique donnée, mais ne s'occupent guère des tactiques de plus haut niveau utiles à la résolution, telles que méthodes multigrille.

Ces outils génèrent en général directement des sous-programmes dans le langage cible (*e.g.* FORTRAN). C'est à notre avis une grande faiblesse. En effet la synthèse doit alors s'occuper de deux problèmes à la fois, qui sont la nature de l'algorithme à implémenter, et la manière d'implémenter cet algorithme (langage, noms de variables, étiquettes, parenthèses...). Or il s'agit de deux problèmes clairement séparés, et leur mélange conduit à des outils lourds et difficilement maintenables.

2.3 Avantages sur les équipes concurrentes

D'autres équipes dans le monde travaillent sur la différentiation automatique. Elles correspondent principalement aux outils de différentiation existants. ADIFOR est développé à Argonne (Paul Hovland et Lucas Roh) et à Rice (Alan Carle). ADOL-C est le support des recherches à l'université de Dresde (Andreas Griewank). L'équipe de l'université de Lyngby (Bendsten et Stauning) développe l'outil FADBAD. PADRE2 sert aux études sur les erreurs d'arrondi à l'université Chuo de Tokyo (Koichi Kubota). D'autres équipes font des recherches sans pour autant proposer d'outil, comme à l'université du Hartfordshire (Bruce Christianson et Steve Brown), au GKSS de Hambourg (Andreas Rhodin), ou à Cornell (Tom Coleman et Arun Verma).

En parallélisation, le nombre d'équipes est beaucoup plus grand. Pour la parallélisation d'instructions et de boucles, le sujet en vogue est OPENMP. Un autre sujet brûlant est l'optimisation des accès à la mémoire, quand elle est distribuée, ou vue au travers de caches. Pour la parallélisation SPMD qui nous intéresse, les équipes proches sont le projet ARCHIMEDES à Carnegie-Mellon (Dave O'Hallaron et Jon Shewchuk) ou le projet DIME à CalTech (R.D. Williams). Mais notre méthode de parallélisation, développée par le projet SINUS, paraît plus élégante.

Ce qui doit nous permettre de différencier notre travail par rapport à ces équipes consiste en trois points principaux:

i Nous avons une compétence dans plusieurs classes d'analyses et transformations de programme. Principalement la différentiation automatique, la parallélisation et l'évaluation partielle. Cela nous permettra de transposer les expériences entre ces classes.

ii L'expérience de CENTAUR donne une approche plus rationnelle des outils d'analyse de programme, en particulier en découplant les problèmes. Par exemple cette expérience montre que l'on ne doit pas mélanger des problèmes syntaxiques et sémantiques.

iii L'expérience de PARTITA nous a montré l'intérêt de fonder les analyses et les transformations sur le graphe de flot. Cela simplifie les structures de contrôle que l'on doit traverser lors des analyses par interprétation abstraite.

3 Aspect Recherche

3.1 Acquis Scientifique des membres du projet

Les membres du projet TROPICS ont acquis une solide expérience dans le domaine des outils de transformation semi-automatiques de programmes en général, et en particulier appliqués à la Différentiation Automatique ou la parallélisation. Ils ont également acquis une bonne connaissance des problèmes spécifiques liés au calcul scientifique, et des méthodes et outils théoriques utilisés dans ces domaines. Cette connaissance est nécessaire pour effectuer des recherches fructueuses avec les acteurs du calcul scientifique, proposer des outils réalisables qui répondent à leurs attentes, et comprendre et tenir compte des retours d'utilisation.

D'un point de vue général, les analyses et transformations de programmes constituent l'expérience commune des membres du projet. Valérie Pascual et Laurent Hascoët ont été membres du projet CROAP. Depuis 1984, ce projet a exploré diverses voies pour réaliser ces outils, en particulier par l'utilisation de règles d'inférence logique, une méthode connue sous le nom de *sémantique naturelle*. Ces travaux ont mené à la construction de l'outil CENTAUR, un générateur d'environnements de programmation, qui constitue une expérience de première importance dans le domaine de la spécification et réalisation d'outils de manipulation de programmes. Dans le cadre de CROAP, Valérie Pascual a travaillé sur des outils génériques d'analyse et de transformation de programme, fonctionnalités essentielles de CENTAUR. Elle a étudié particulièrement les outils interactifs de manipulation de programmes [9], et des environnements de programmation pour JAVA. Au sein du projet CROAP, Laurent Hascoët a travaillé sur des questions concernant l'interface utilisateur de CENTAUR (chemins d'arbres [29], afficheur incrémental FIGUE). Il a étudié également plusieurs types d'outils sémantiques, dont un vérificateur de types incrémental, [61] et un évaluateur partiel [62] dont il a dérivé un outil de manipulation formelle d'arbres de preuve [60]. Également, Christèle Faure a travaillé, dans le domaine du calcul formel, sur les questions de simplification [38], importantes dans les transformations de programme.

En ce qui concerne la Différentiation Automatique, l'expérience du projet provient des travaux du projet SAFIR, depuis 1992. Ce projet a publié des recherches concernant la complexité théorique des programmes différentiés [75], et des compromis entre stockage et recalcul pour le mode inverse, dont l'optimalité a été prouvée dans [59]. Ces travaux ont mené à la construction de l'outil ODYSSÉE, un outil de Différentiation Automatique qui jouit d'une reconnaissance internationale, marque évidente de sa qualité scientifique. Au cours de ces deux dernières années, Christèle Faure a travaillé tout particulièrement sur le mode inverse et ses difficultés. De l'applicabilité de ce mode dans un cadre industriel dépend en grande partie la diffusion de cette nouvelle technologie. Les contrats successifs avec EDF-DER qu'elle a coordonnés montrent la faisabilité de la méthode (même sur un code de 60 000 lignes), mais de nombreuses optimisations sont encore nécessaires pour que cette technique soit largement utilisée. De plus, Christèle Faure coordonne l'action de recherche coopérative "Mode inverse opérationnel" (MIO). Cette action étudie les optimisations à appliquer au code généré pour atteindre les performances des codes adjoints écrits à la main. En amont, ces optimisations nécessitent des analyses statiques du programme, pour lesquelles l'expérience des membres du projet sera un atout essentiel. Concernant les calculs de matrices jacobiennes, Mohamed Tadjouddine a effectué une recherche concernant la détection automatique de certaines jacobiennes creuses. Il a utilisé des méthodes et outils d'analyse statique des programmes, provenant de la parallélisation [93]. Il a soutenu sa thèse en Mars 99. Ses travaux constituent une piste de recherche intéressante. Dans la perspective des développements futurs d'ODYSSÉE, Valérie Pascual et Laurent Hascoët travaillent actuellement sur une architecture plus distribuée, et sur les protocoles à mettre en œuvre pour faire communiquer des processus indépendants, (analyse syntaxique, analyses sémantiques statiques, synthèse du programme différentié...)

En ce qui concerne la parallélisation, le savoir-faire du projet provient de l'expérience de la société CONNEXITÉ, dont Laurent Hascoët a été cofondateur en 1992. S'inspirant de l'outil PIAF [48], la société a construit le paralléliseur interactif PARTITA, qui a ensuite été repris par la société SIMULOG. Dans le cadre de CONNEXITÉ, puis pour SIMULOG, Laurent Hascoët a étudié en détail les outils de génie logiciel pour le calcul scientifique. Pour la parallélisation, le développement de l'outil de parallélisation PARTITA a nécessité la construction de plusieurs algorithmes originaux [63]. En particulier, les analyses sont basées sur le graphe de flot de contrôle, et utilisent des principes provenant de l'interprétation abstraite [31] pour la détection des variables d'induction, l'analyse statique des pointeurs, le calcul des dépendances, et la détection et la régénération des parties parallèles. Dans le cadre du projet SINUS, Laurent Hascoët a étudié l'automatisation partielle d'une méthode de parallélisation particulière (partition de maillage [64], [65] prototype GASPARD). Ce travail l'a conduit à se familiariser avec les problèmes et les théories du calcul scientifique.

Concernant les directions de travail plus prospectives, Laurent Hascoët a encadré une étude sur la synthèse automatique de parties de programmes à partir d'équations aux dérivées partielles [88], et il a dirigé la thèse de Laurent Angeli [6], portant sur la détection des similitudes entre deux fragments de programme, dans l'optique d'un outil d'aide à la maintenance de gros programmes.

3.2 Programme de Recherche

Nous considérons qu'il y a une unité manifeste parmi les outils de manipulation de programmes scientifiques que nous avons listés plus haut. Par conséquent, nous présenterons d'abord les directions de recherche suivantes dans ce cadre général:

- **a)** Analyses inter-procédurales, et en particulier la propagation, la spécialisation et la généralisation.
- **b)** Analyses intra-procédurales, représentation interne abstraite.

Cependant, les deux transformations centrales, D.A. et parallélisation, posent aussi des problèmes de recherche spécifiques. Nous identifions les sujets suivants. Pour la Parallélisation:

- **c)** Parallélisation vers OpenMP
- **d)** Parallélisation SPMD
- **e)** Alignements HPF
- **f)** Problèmes de mémoire distribuée

Pour la Différentiation:

- **g)** Performances calcul et mémoire
- **h)** Mode "fast forward"
- **i)** Lien avec le parallélisme
- **j)** Matrices jacobiennes creuses
- **k)** Exploitation des cas particuliers
- **l)** Nouveaux modes
- **m)** Voisinage de validité d'une différentiation

Enfin, d'une manière plus prospective, nous gardons à l'esprit deux thèmes de recherche,

- **n)** la synthèse de parties de programmes.
- **o)** la reconnaissance et la comparaison de sous-programmes.

3.2.1 Recherches d'intérêt plus général

a) Analyses inter-procédurales

Les manipulations de programme nécessitent souvent des analyses préliminaires, et ce sont en général les mêmes. Outre la vérification statique des types, citons diverses propagations (constantes, contraintes, variables actives de dérivation...), et analyses (pointeurs, alias, lectures/écritures, dépendances). Ces analyses – qui sont souvent mutuellement dépendantes – n'ont d'intérêt réel que dans la mesure où elles sont interprocédurales. Cela oblige à réfléchir à l'ordre entre les analyses des divers sous-programmes. En particulier pour des analyses *descendantes* le long du graphe d'appel, on peut rencontrer une même routine plusieurs fois, ce que l'on sait gérer soit en spécialisant la routine pour chaque cas d'appel, soit en généralisant les cas d'appel en en effectuant l'*union*. Ces choix ont été étudiés en partie pour l'évaluation partielle [17]. Peut-on proposer une approche plus systématique, qui aurait de bonnes propriétés, de complexité par exemple? Que se passe-t-il dans le cas de programmes récursifs? Cela réclame aussi une réflexion sur des supports communs pour spécifier et implémenter efficacement un outil interprocédural quelconque. Dans le cadre de cette recherche, il importe également d'étudier l'aspect incrémental, aux niveaux intra- et inter-procédural. L'incrémentalité des analyses est nécessaire pour une application à des programmes en cours de développement.

b) Analyses intra-procédurales

Les manipulations opèrent sur une représentation interne des programmes. On doit chercher une forme plus versatile et abstraite que les arbres syntaxiques. Cette représentation devrait être adaptée aux différentes manipulations, tout en fournissant un certain confort pour l'implémentation. De plus, en se restreignant aux langages impératifs, il semble naturel d'exploiter leurs grandes similitudes dans une forme commune. L'expérience des compilateurs à forme intermédiaire commune (GCC...) peut servir de guide. Le fait que les évolutions récentes de la plupart des langages impératifs aillent dans les mêmes directions (pointeurs et allocation dynamique, structures de données et objets, vectorisation et parallélisation...) plaide également dans le sens d'une forme intermédiaire commune. Parmi les modes de représentation classiques, on considèrera les arbres syntaxiques, mais aussi les graphes de flot de contrôle, structurés ou non, ou encore les "program dependence graphs" [84] [86]. Il faut aussi chercher une représentation se prêtant bien à la synthèse de programmes.

3.2.2 Parallélisation

c) Parallélisation vers OpenMP

La parallélisation est un domaine très vaste, et il faut donc bien préciser les directions de recherche. En ce qui concerne les méthodes de parallélisation généralistes, nous ne prévoyons pas de sujet urgent. Cependant, nous surveillons l'évolution de la tentative OPENMP [82], qui nous permettrait de faire fructifier le travail déjà effectué dans PARTITA (parallélisation fine de boucles), moyennant quelques travaux autour de possibilités spécifiques d'OPENMP.

d) Parallélisation SPMD par échange de messages

Pour le reste, nous considérons que l'on doit orienter les recherches vers des méthodes et outils de parallélisation spécialisés, pour certaines classes de programmes. La première classe identifiée, les calculs itératifs sur maillages irréguliers, se prête bien à une parallélisation SPMD par échange de messages. Il faudrait voir si cette méthode s'étend aux méthodes multigrille, ou aux maillages adaptatifs. De plus, et en anticipant sur les paragraphes suivants, nous devons étudier les liens entre cette parallélisation et la D.A.

e) Alignements hpf

Par ailleurs, il faut identifier d'autres classes, et leurs méthodes de parallélisation spécifiques, ce qui ne peut se faire qu'en concertation avec les numériciens. À la limite, nous considérons que la parallélisation vers HPF est une méthode spécifique: pour le cas des programmes très réguliers, tels que les bibliothèques d'algèbre linéaire. C'est sur cette classe que HPF donne de bons résultats, et non pas sur des programmes utilisant des maillages irréguliers. Dans ce cadre, un sujet actif est le choix semi-automatique des directives d'alignement. Pour cela, on recherche d'une part les boucles parallélisables, ce qui est en grande partie automatisable, et d'autre part on aligne les tableaux par rapport aux processeurs parallèles, ce qui est beaucoup moins automatisable. Il reste ici de nombreux problèmes pour obtenir des algorithmes plus efficaces, et concevoir une interaction intelligente avec l'utilisateur.

f) Problèmes de mémoire distribuée

Un autre sujet intéressant est l'optimisation des programmes pour la gestion de la mémoire cache. Nous ne sommes pas suffisamment compétents sur ce sujet, dont d'autres projets à l'INRIA sont déjà spécialistes. Mais nous entendons collaborer avec eux pour leur fournir des outils d'analyse. En fait, cette collaboration existe déjà, avec l'utilisation de PARTITA comme une base d'implémentation dans le projet CAPS à l'IRISA.

3.2.3 Différentiation

g) Performances calcul et mémoire

Un premier thème concerne la consommation mémoire et le calcul des programmes différentiés. On a vu que le mode inverse utilise les valeurs intermédiaires du programme dans l'ordre inverse de leur production. Il faut donc des moyens pour que ces valeurs soient disponibles. Le moyen le plus simple est le stockage, mais le coût en mémoire est vite prohibitif. On peut tout d'abord limiter la quantité de valeurs stockées des manières suivantes:

- Par une analyse globale, détecter les valeurs intermédiaires non utilisées, ou du moins pas dans la dérivée (expressions linéaires)
- Recalculer au lieu de stocker, lorsque c'est possible. Par exemple en dupliquant l'instruction simple qui calculait cette valeur, ou en inversant une instruction simple qui utilisait cette valeur.

- Pour le nécessaire stockage du flot de contrôle, on peut mémoriser les numéros des "basic blocks" parcourus, ou directement "inverser" le graphe de flot de contrôle. Peut-on comparer ces méthodes? Peut-on inverser tous les graphes de flot, et à quel prix?

Cela ne suffisant pas en général, on doit accepter d'oublier les valeurs intermédiaires sur certaines portions de l'exécution (entre des points appelés "checkpoints"), pour ensuite exécuter à nouveau ces portions plus tard, cette fois-ci en stockant les valeurs intermédiaires. Cela amène à des compromis stockage/recalcul, qui ont été très bien étudiés dans un cas. Il s'agit des programmes constitués d'une boucle principale, dont le nombre d'itérations est connu, et chaque itération considérée comme atomique pour les checkpoints. Il serait très important pour des programmes réels d'étudier le cas très fréquent où le nombre d'itérations n'est pas connu. En allant plus loin, les checkpoints ne devraient pas être cantonnés aux débuts d'itérations de la boucle principale. Par exemple, dans un mode particulier, ODYSSÉE relance l'exécution de certains sous-programmes plusieurs fois, ce qui s'apparente à un mécanisme de checkpoints. Cette manière est commode, mais il n'est pas certain qu'elle soit optimale. Quels sont les points de checkpoint possibles, et quelle est une bonne répartition de ces points? Nous devons trouver un cadre commun pour comparer les différentes manières de placer les checkpoints.

En un checkpoint donné, quel est l'ensemble minimal des valeurs nécessaires pour pouvoir relancer l'exécution jusqu'au checkpoint suivant? Cet ensemble diffère bien entendu d'un point à l'autre, et cela doit être pris en compte lors du choix des checkpoints. Dans certaines applications, cet ensemble correspond aux "variables d'état", dont les autres sont déduites. Ce point est étudié dans le cadre de l'action MIO. Signalons en outre une direction de recherche originale, proposée par A. Griewank: on pourrait utiliser l'"inversion" d'un programme pour revenir en arrière en un point d'une exécution, par exemple à l'intérieur d'un outil de debug.

h) Mode "fast forward"

Une question assez proche a été proposée par A. Griewank. En mode direct et inverse, il arrive qu'on sache que le programme différentié va être appelé un certain nombre de fois, pour un même ensemble des valeurs d'entrée, mais pour des directions de différentiation différentes. Cela arrive par exemple pour la méthode dite "Newton tronqué". Il peut être intéressant de repérer les calculs qui sont à la fois coûteux et indépendants de la direction de différentiation. On pourrait alors les conserver sous forme de mémo-fonctions (mode "fast forward"). Un autre problème lié est de repérer les sous-expressions communes dans le programme dérivé. Il ne s'agit pas ici de les détecter dans des expressions a priori quelconques (problème NP-complet), mais de ne pas laisser la D.A. introduire de *nouvelles* expressions communes. Les premières expériences laissent espérer des performances supérieures, dans certains cas, aux différences divisées, une méthode manuelle, "concurrente" principale de la D.A. en mode direct.

i) Lien avec le parallélisme

Une deuxième classe de problèmes concerne le lien avec le parallélisme. Il y a deux directions principales, indépendantes. Tout d'abord, étudier comment la D.A. peut conserver les propriétés de parallélisme du programme initial. Ces questions sont particulièrement délicates dans le cas du mode "inverse" de différentiation. Par exemple, à quoi ressemble la différentiation en mode inverse d'un programme vectorisé? Ou encore, et c'est le sujet d'une étude financée par le programme GÉNIE, quand ce programme initial est parallélisé selon une approche SPMD, avec mémoire distribuée et échange de messages. D'autre part, une question plus récente est comment générer des programmes différentiés plus efficaces, qui utiliseraient les capacités d'une machine parallèle. On imagine que chaque processeur peut gérer un certain nombre de checkpoints, mais on n'a pas encore d'idée claire sur une manière optimale.

j) Matrices jacobiennes creuses

La différentiation en vue du calcul des matrices jacobiennes complètes pose des problèmes spécifiques. En particulier, comment détecter et exploiter le fait que ces jacobiennes sont souvent creuses? Le travail de thèse de Mohamed Tadjouddine présente une réponse lorsque les programmes sont réguliers (accès réguliers aux tableaux). Mais les programmes sont souvent irréguliers. Que peut-on détecter alors? On peut analyser la forme creuse des jacobiennes par une exécution "observée" du programme, ce qui se rapproche des méthodes "inspector-executor" utilisées en parallélisation. Sinon, peut-on exploiter une connaissance plus fine donnée par l'utilisateur? Par exemple, pour des maillages, la géométrie du maillage pourrait donner la structure de la matrice jacobienne. Il semble en tout cas que pour ces questions, il soit vite nécessaire de se restreindre à des classes particulières de programmes scientifiques. Aussi, pour limiter la taille des jacobiennes, on peut ne demander que les dérivées de certaines sorties par rapport à certaines entrées. Cela est très proche de l'évaluation partielle (limitation par rapport aux entrées) et du slicing (limitation par rapport aux sorties), et les méthodes employées dans ces domaines doivent être transposables chez nous.

k) Exploitation des cas particuliers

La D.A., comme la parallélisation, gagne à exploiter les propriétes particulières de certains (fragments de) programmes. Par exemple, il est probable qu'on pourra obtenir une meilleure performance si l'on reconnaît qu'un fragment implémente une décomposition LU. On voit le lien avec la reconnaissance de fragments de programme. À un niveau plus global, des études théoriques montrent que l'adjoint (mode inverse) de certaines résolutions itératives (point fixe) ne nécessite pas la mémorisation des premières itérations, dans le cas où seul compte le résultat à la dernière itération [51], [27]. Peut-on caractériser d'un point de vue informatique les programmes sur lesquels cette remarque s'applique, ou bien doit-on s'en remettre à une directive fournie par l'utilisateur? Y a-t-il d'autres catégories de programmes qu'il serait bon de traiter différemment?

l) Nouveaux modes

De nouveaux modes de différentiation sont peut-être envisageables. Il existe des recherches sur des combinaisons des modes direct et inverse. Par endroits, sur des parties de programme bien particulières, certains pensent à précalculer des matrices jacobiennes partielles. Certaines études tentent d'optimiser le calcul des dérivées en raisonnant sur le *graphe de calcul*, qui représente l'ensemble des calculs nécessaires à cette dérivée. Peut-on en dériver de nouveaux modes de différentiation plus efficaces, sans qu'il soit nécessaire de construire cet *énorme* graphe de calcul?

m) Voisinage de validité d'une différentiation

Il reste à étudier les "limites" de la D.A. Quel rapport y a-t-il entre la dérivée formelle et le programme différentié? Il est bien connu que la D.A. voit les programmes comme dérivables par morceaux. Que se passe-t-il à la frontière entre les morceaux? Bien que le programme n'y soit pas différentiable en toute rigueur, on peut s'intéresser aux dérivées directionnelles (*e.g.* à droite ou à gauche), qui elles, peuvent être définies. Au minimum, il est bon que le programme différentié puisse avertir l'utilisateur du risque d'invalidité de son résultat. Un cas analogue est celui des fonctions implicites résolues par dichotomie: informatiquement, le programme décrit une fonction constante par morceaux, et donc la dérivée calculée sera nulle, ce qui n'est pas le résultat attendu. Peut-on détecter ce cas de figure, par exemple en calculant, en même temps que la dérivée, une approximation de son domaine de validité. Dans le cas de la dichotomie, ce domaine serait petit comme la précision machine, ce qui devrait alerter l'utilisateur final.

Il faut enfin mentionner que la différentiation automatique présente un intérêt pour l'estimation des erreurs d'arrondi d'un programme. Cette voie est explorée par exemple dans une thèse à l'Université de Nice, en utilisant certaines bases communes avec ODYSSÉE. Cela a donné lieu à l'écriture d'un nouveau logiciel ODYPER.

3.2.4 Directions prospectives

n) Synthèse de parties de programmes

La synthèse de parties de programmes scientifiques à partir d'équations est peut-être la direction la plus prospective. Les programmes de calcul scientifique représentent de gros investissements en programmation, dans les milieux académique et surtout industriel. La maintenance de ces programmes nécessite un effort permanent. Il est par exemple fréquent de changer la modélisation physique du problème (par exemple ajout d'inconnues et d'équations pour rendre compte d'une réaction chimique). Il est encore plus fréquent de modifier le schéma de discrétisation choisi (schémas décentrés, ou approximations d'ordre supérieur). On peut aussi changer le type des mailles de la discrétisation, ou combiner des mailles de formes différentes. Ces modifications entraînent la réécriture de la partie centrale du programme. D'où l'idée de tenter d'automatiser l'écriture d'une partie de ces programmes, à partir d'une forme source à définir. Un changement pourrait alors se traduire par une modification de cette source. Les expériences antérieures montrent qu'il n'est pas réaliste actuellement, de vouloir générer l'ensemble du programme, mais plutôt de générer uniquement certaines parties bien isolées. Pour la mécanique des fluides, on pense en premier lieu à la partie centrale, qui calcule les "flux" discrétisés sur les éléments du maillage. Les données seraient les équations aux dérivées partielles, et les schémas de discrétisation. Le résultat serait un module insérable dans le programme final. Ce dernier serait responsable des tactiques de plus haut niveau, telles que adaptation du maillage, ou algorithmes multigrille. Ces tactiques ne sont pas liées aux équations physiques de départ, et semblent pour l'instant trop complexes pour être du ressort d'un outil de synthèse automatique. La synthèse permet de plus la mise en œuvre de méthodes de D.A. et parallélisation spécifiques, tenant compte d'une connaissance du problème initial que l'on ne peut déduire du programme seul, sauf parfois en cherchant des similitudes avec des routines-type.

o) Reconnaissance et comparaison de sous-programmes

L'étude des similitudes entre fragments de programme a été défrichée dans [6]. Elle peut être améliorée. Il reste aussi un travail important pour définir précisément la relation d'équivalence que l'on peut reconnaître partiellement. Par "partiellement", nous entendons que ce problème étant en général indécidable, on ne peut pas chercher un réponse tranchée dans tous les cas, et on devra accepter, outre "oui" et "non", une troisième réponse "peut-être". Cette remarque s'applique en fait à la plupart des analyses statiques, et nous la sous-entendrons dans la suite. Cela étant, de nombreuses applications de la reconnaissance de similitudes paraissent intéressantes. Nous avons déja noté que la reconnaissance d'opérations mathématiques classiques peut faciliter la D.A. (*cf* 3.2.3 **k)**), la parallélisation, ou la documentation/compréhension d'un code. Il y a là un lien avec l'objectif ambitieux du projet CABARET (TSF) à l'IRISA. Il y a enfin des études théoriques intéressantes à poursuivre sur la sélection automatique des fragments à comparer, qui se heurte à d'évidents problèmes d'explosion combinatoire, et des recherches pour améliorer et prouver les algorithmes de comparaison.

4 Aspect Développement Logiciel

4.1 Acquis Technique

L'environnement CENTAUR, développé par le projet CROAP, fournit un cadre pour spécifier et implémenter des analyses et transformations de programme. À ce titre, il contient un ensemble d'outils fondamentaux:

i Un générateur d'analyseur syntaxique.

ii Une représentation interne des arbres syntaxiques.

iii Un langage de spécification d'analyses par parcours de l'arbre syntaxique.

iv Une interface utilisateur générique, utilisant une sélection structurée.

v Un générateur d'afficheur.

Nous nous appuierons sur les points *i*, *ii* et *v*, qui constituent un acquis technique important.

L'outil PARTITA, bâti sur CENTAUR, reste une base pour l'implémentation des recherches en parallélisation. Il fournit une analyse de dépendances qui a fait ses preuves sur de très nombreux programmes industriels, ainsi que des analyses préliminaires (pointeurs, variables d'induction,...) indispensables à une bonne analyse des dépendances. Toutes ces analyses sont implémentées sur le graphe de flot de contrôle. Il représente également une expérience en vraie grandeur pour la régénération de programmes, en suivant la chaîne inverse, qui va d'un nouveau graphe de flot de contrôle à un nouvel arbre syntaxique [11]. L'outil PARTITA proprement dit permet la parallélisation en direction de FORTRAN95, HPF, ou d'autres dialectes parallèles de FORTRAN. Il a en outre servi de base pour l'implémentation de GASPARD, spécialisé dans la parallélisation SPMD de programmes sur des maillages. Les analyses de PARTITA ont été utilisées pour d'autres travaux:

- Un interfaçage avec le paralléliseur de boucles BOUCLETTE, du LIP [22].
- L'implémentation des algorithmes de comparaison de programmes de la thèse de Laurent Angeli [6].
- L'étude des optimisations de cache dans le cadre du projet CAPS.

- La détection statique de certaines matrices jacobiennes creuses [93].

ODYSSÉE est l'un des principaux outils actuels de D.A. Il génère des programmes dérivés soit en mode direct (linéaire tangent), soit en mode inverse. À la différence d'ADOL-C, qui surcharge les opérateurs arithmétiques, ODYSSÉE construit un nouveau programme qui calcule les dérivées. Dans le cadre de contrats industriels, ODYSSÉE a déjà été utilisé sur de nombreux codes réels. On peut considérer que ses principes sont ainsi validés. Le mode inverse, malgré ses défauts de jeunesse, est une excellente base pour tester les idées et recherches décrites dans la section 3.2.3 **g)**. Il faut noter qu'une nouvelle version de ce mode inverse est en cours de validation. Cette version s'appuie sur une transformation du graphe de flot de contrôle, et utilise un nouveau mode de stockage des valeurs intermédiaires. Elle permettra d'appliquer l'outil à des programmes plus nombreux.

4.2 Programme de Développement

L'applicabilité pratique des recherches étant primordiale à notre avis, chaque étude devrait aboutir au moins à un prototype. Il sera alors nécessaire de disposer d'une base d'implémentation commune pour ces prototypes, indépendamment de l'outil particulier. Ainsi, même si parfois, les études théoriques s'éloignent du cadre général des transformations de programme, l'utilisation d'une base commune pour les prototypes souligne les liens entre ces travaux.

Dans un premier temps, les prototypes associés aux recherches seront probablement implémentés dans les systèmes ODYSSÉE ou PARTITA actuels.

- ODYSSÉE pour les prototypes concernant la différentiation automatique.
- PARTITA pour les prototypes utilisant l'analyse des dépendances, tels que parallélisation ou la comparaison de sous-programmes.

L'essentiel des recherches portera d'abord sur la D.A. Il accompagnera un effort de développement plus poussé sur ODYSSÉE. Dans le même temps, des développements importants d'ODYSSÉE auront lieu dans le cadre de l'action coopérative MIO, et de l'action de développement DIFFPRO. Il est bien entendu que tous ces développements, poursuivant des objectifs proches, seront étroitement coordonnés. Sans définir de priorités, les développements supplémentaires à apporter à ODYSSÉE pourraient être:

- lever certaines restrictions au sous-langage de FORTRAN accepté par ODYSSÉE. Avertir automatiquement des restrictions restantes.

- différentier en modes direct et inverse du FORTRAN90, en traitant correctement la notation vectorielle.

- générer un mode inverse par une inversion générale du graphe de flot de contrôle.

- minimiser la quantité de valeurs stockées, tant pour celles utilisées par la phase *inverse*, que celles qui permettent de relancer l'exécution à partir d'un *checkpoint*.

- accepter des directives de l'utilisateur, qui indiquerait un meilleur schéma de *checkpointing*, ou des fragments de programme permettant une différentiation particulièrement efficace.

- donner des outils de validation des programmes différentiés.

À plus long terme, il faudra rapprocher les deux systèmes ODYSSÉE et PARTITA. Une raison parmi d'autres, est que la D.A. profiterait d'une analyse des dépendances, pour limiter les stockages en mode inverse, ou pour déterminer la structure creuse des matrices jacobiennes. On rencontrera alors un problème technique, qui est la différence entre les représentations internes, ainsi que la différence des langages d'implémentation (Le-Lisp contre Caml). Il est donc très important de réfléchir tout de suite à ces questions d'architecture. Il ne semble pas réaliste, surtout pour un projet de recherche, de traduire d'emblée l'un des systèmes dans le langage de l'autre. La solution semble être dans une plus grande modularité, dans laquelle des "processus" séparés s'occuperaient de

- l'analyse syntaxique

- l'affichage et l'interface utilisateur éventuelle

- l'analyse de dépendances

- la différentiation

- la parallélisation

en communiquant par des protocoles à définir (dès maintenant). Les développements futurs nous amèneront alors à (ré-)écrire certains modules, sans que cela affecte trop le fonctionnement des autres modules. L'un de nos objectifs à court terme (minimisation du nombre de valeurs stockées par le mode inverse), nécessitant des analyses nouvelles, basées sur le graphe de flot, le premier module que nous écrirons sera un analyseur de sous-programmes, travaillant sur leur graphe de flot. Nous envisageons pour ce module une implémentation en JAVA. La figure 1 illustre ce découpage. Cette définition

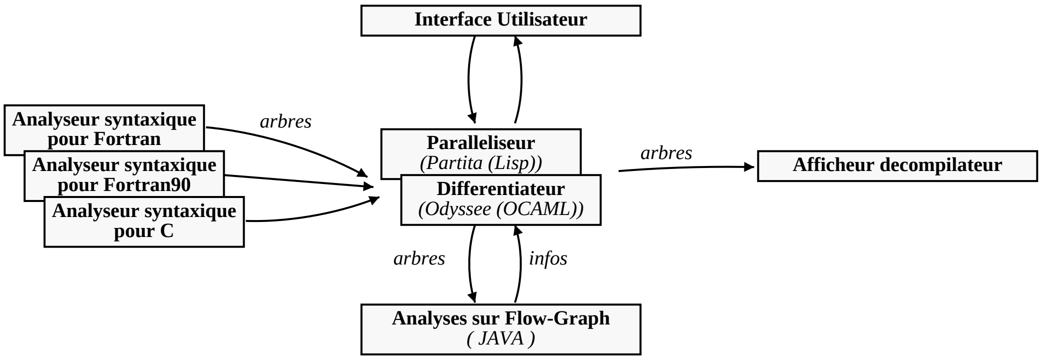

Figure 1: *Coopération entre process, pour* ODYSSÉE *et* PARTITA

des modes de communication est en cours, et nécessite certainement un travail scientifique. À terme, nous disposerons d'un environnement commun pour la description et l'implémentation de diverses transformations de programmes scientifiques. Cela permettra le prototypage de nouveaux outils pour numériciens, comme décrit plus haut dans la section 2.1.3.

5 Programme de travail

À moyen terme, il va falloir choisir entre ces nombreuses directions. Les problèmes liés à la D.A. auront priorité dans une première phase d'environ 3 ans. Aujourd'hui, il nous semble raisonnable de faire porter l'effort de recherche à moyen terme sur les sujets suivants. Pour chaque direction, nous précisons les membres du projet a priori les plus concernés.

- Les analyses intra-procédurales, et la représentation interne abstraite. *cf* 3.2.1 **b)**. *(participants: Valérie Pascual, Laurent Hascoët, Frédéric Olier)*

- La parallélisation SPMD dans ses interactions avec la D.A. *cf* 3.2.2 **d)** et 3.2.3**i)**. *(participants: Christèle Faure, Patrick Dutto, Stefka Fidanova, Laurent Hascoët)*

- Les performances calcul et mémoire des programmes différentiés. *cf* 3.2.3 **g)**. *(participants: Christèle Faure, Laurent Hascoët, Uwe Naumann)*

- les modes nouveaux ("fast forward" ou autres). *cf* 3.2.3 **h)** et **l)**. *(participants: Andreas Griewank, Christèle Faure, Uwe Naumann, Laurent Hascoët)*

Nos critères de réussite correspondent à ces quatre directions, auxquelles nous ajouterons la réalisation d'une architecture d'Odyssée correspondant au schéma de la figure 1. Les autres sujets de recherche cités ne démarreront probablement pas avant deux ans. Précisons que notre collaboration avec Andreas Griewank continuera après son retour à Dresde. Nous en prendrons pour preuve la co-organisation de la conférence "AD2000". Il se pose néanmoins un grave problème d'effectifs, dont nous sommes conscients. Nous faisons tout le nécessaire pour que, après le départ de Christèle Faure en 8/2000, le projet puisse continuer sans heurts. Si nos effectifs restent alors a deux permanents, il sera nécessaire de recentrer nos objectifs, en privilégiant encore une fois la D.A. Mais le maintien de l'activité du projet nécessitera probalement un nouveau recrutement.

6 Collaborations, Diffusion et Valorisation des Résultats

6.1 Compétences extérieures nécessaires au projet

Il ne fait pas de doute que l'étude d'outils informatiques pour les numériciens réclame le rapprochement d'au moins trois types de compétences:

i La connaissance des problèmes rencontrés par les numériciens, et de leurs outils théoriques habituels.

ii La connaissance des méthodes et techniques de manipulation de programmes.

iii La connaissance algorithmique pour concevoir et évaluer les algorithmes d'analyse et transformation. La complexité de ces analyses doit être étudiée de près, ainsi que celle des programmes résultats.

Un projet a une "culture". La nôtre est principalement du type *ii)*, et partiellement du type *iii)*. C'est là que nous comptons apporter quelque chose. Pour les compétences du type *i)*, nous interagirons en particulier avec les projets de numériciens, SINUS, CAIMAN, IDOPT, ESTIME (Jean-Charles Gilbert). Nous pensons posséder de bonnes bases du domaine *iii)*. Néanmoins, nous n'hésiterons pas à coopérer avec des spécialistes, que nous trouverons dans les projets MIAOU, IDOPT, ALGO, mais aussi avec Andreas Griewank après son retour à Dresde.

6.2 Positionnement par rapport aux autres projets inria

L'activité de D.A., ainsi que l'outil ODYSSÉE, proviennent du projet SAFIR, en collaboration avec l'université de Nice. Plusieurs anciens membres de SAFIR s'intéressent toujours à la D.A., et nous souhaitons continuer la collaboration.

D'autre part, dans le cadre de l'action coopérative de recherche MIO, financée par l'INRIA, nous avons établi des liens avec les projets IDOPT et PARA. La collaboration avec Isabelle Charpentier, du projet IDOPT, suggère des améliorations à apporter à ODYSSÉE, du point de vue de l'utilisateur numéricien, mais aussi des idées sur la marche à suivre. La collaboration avec PARA apporte des éclairages théoriques sur les transformations de programme.

Le projet CROAP a créé l'environnement CENTAUR, sur lequel est bâti PARTITA. Ce projet et ses descendants, restent à la pointe pour les outils d'analyse et de transformation de code. La collaboration avec les projets successeurs de CROAP sera évidemment maintenue.

Nous voulons aussi développer les liens avec le projet CAPS en particulier avec François Bodin et Yann Mevel, pour leur utilisation de certains composants de PARTITA, et leur connaissance des questions de gestion mémoire pour les machines parallèles. En parallélisation, l'expérience du projet APACHE nous intéresse aussi. Nous prévoyons des actions communes avec le projet A3 (commun INRIA et Université de Versailles-Saint-Quentin), qui s'occupe de questions proches, et dont certains chercheurs étaient déjà à l'origine du système PIAF, dont PARTITA s'est inspiré. Les projets LANDE et PARA nous semblent également très proches, en tant que spécialistes des analyses de programmes par des méthodes d'interprétation abstraite.

Du projet COMPOSE, nous voulons apprendre des tactiques d'analyse et transformation interprocédurales. Leur expérience des analyses de C nous intéresse aussi.

Nous pensons que nos connaissances en algorithmique et complexité nous permettront de faire avancer notre travail. Cependant, pour des problèmes particulièrement ardus, nous sommes conscients de l'existence à l'INRIA de projets pointus, comme ALGO.

Tant pour la D.A. que pour les autres activités, l'interaction avec les acteurs du calcul numérique est essentielle. À ce titre, nous avons besoin de maintenir une collaboration étroite avec les projets SINUS à Sophia, IDOPT à Grenoble, M3N et ESTIME à Rocquencourt, en particulier avec Jean-Charles Gilbert.

6.3 Collaborations académiques et diffusion

L'outil ODYSSÉE résulte d'une collaboration entre le projet SAFIR et l'Université de Nice. Même si l'activité dans ce domaine à Nice a sensiblement décru, elle demeure intéressante. En particulier, des chercheurs de l'Université de Nice explorent une approche différente de l'inversion du graphe de flot pour la D.A. en mode inverse, dans leur version d'ODYSSÉE. Nous souhaitons la continuation de la collaboration. Cela a déjà bien commencé, par des discussions techniques approfondies avec Serge Fantino, de l'équipe de Frédéric Eyssette à l'université de Nice. Nous ferons en particulier le nécessaire pour réduire la divergence qui est apparue vers la fin du projet SAFIR, entre ODYSSÉE (SAFIR), et ODYPER (Nice), pour mettre en commun le meilleur des deux approches.

Christèle Faure a organisé et participé à de nombreux colloques internationaux sur la D.A., outils, et applications, qui ont beaucoup contribué à la visibilité internationale d'ODYSSÉE.

Andreas Griewank, de l'université de Dresde, est en visite à l'INRIA Sophia pour un an depuis Août 1998. Le but de son séjour est de renforcer les liens entre son équipe, qui développe l'outil de D.A. ADOL-C, et la nôtre. En particulier, il explore des possibilités de recherche et d'implémentation communes, par exemple autour d'optimisations du mode direct ("fast forward"). Après son départ, il est disposé à continuer la collaboration entre notre projet et son équipe à Dresde.

Uwe Naumann, de l'équipe d'Andreas Griewank, a rejoint notre équipe à partir de Janvier 1999, pour un Post-Doc dans le cadre de MIO. Il travaille sur les améliorations à apporter à ODYSSÉE pour réduire la quantité de stockage nécessaire au mode inverse.

Le projet TROPICS, en collaboration avec l'université de Dresde, organise pour le printemps 2000 la troisième conférence internationale de Différentiation Automatique, après Breckenridge en 1990 et Santa Fé en 1996.

Le projet tente actuellement de mettre en place un consortium (INRIA, EDF, DASSAULT, CEA) pour favoriser la diffusion industrielle d'Odyssée. Parallèlement, une action de développement est à l'étude.

Pour ce qui est de la parallélisation, de nombreux liens ont été initiés autour de PARTITA. À l'origine, ces liens ont été noués par l'intermédiaire des contrats de recherche nationaux ou européens auxquels SIMULOG participait. En particulier, nous avons collaboré avec le VCPC (Vienne Autriche), en particulier avec Barbara Chapman, le LIP (Lyon), en particulier avec Yves Robert, l'Ecole des Mines (Fontainebleau) avec Francois Irigoin, et le projet PRISM de l'université de Versailles, en particulier avec Jean-Francois Collard. Ces collaborations ont apporté un début de reconnaissance de PARTITA. Il est clair qu'elles doivent être maintenues.

6.4 Collaborations industrielles et Valorisation

ODYSSÉE a été développé et utilisé dans le cadre de collaborations industrielles avec ELF, CNES, DASSAULT, ESSILOR, EDF, MÉTÉO-FRANCE et ALENIA. Les liens avec ces compagnies restent forts. Les collaborations industrielles autour d'ODYSSÉE sont actuellement:

- Le projet européen HPCN DECISION, qui construit une plateforme d'outils pour l'optimisation, qui utilise ODYSSÉE.
- Le programme de collaboration GÉNIE, avec DASSAULT, dans lequel on s'intéresse aux liens entre la D.A. et la parallélisation SPMD des programmes numériques.
- Un contrat de recherche et de développement partiellement financé par EDF-DER.
- L'utilisation d'ODYSSÉE par SIMULOG pour des activités de service et de consultant.

De plus, un consortium (et une action de développement INRIA) est en train de se mettre en place autour d'ODYSSÉE, qui regroupera pour commencer EDF, DASSAULT, le CEA et SIMULOG. Le projet TROPICS lui sera naturellement associé.

ODYSSÉE est distribué dans de nombreux sites dans le monde. En France, CEMEF, IFREMER, ORSTOM, PROSIM, ELF, IFP, INRA, ESI-FRANCE, MÉTÉO-FRANCE. À l'étranger, BP (Angleterre), Belgian Institute for Space Aeronomy, Argonne National Laboratory, Los Alamos National Laboratory, National Center for Environmental Prediction (Washington DC), ainsi que les universités de Cologne (Allemagne), Dresde (Allemagne), Coimbra (Portugal), Calabre (Italie), Lodz (Pologne), Rio (Bresil), Marquette (Wisconsin), Arizona State (Arizona), Tulsa (Oklahoma).

SIMULOG commercialise PARTITA (nom commercial FORESYS-EXPERT). Il existe actuellement de l'ordre de 150 jetons FORESYS vendus dans le monde, dont une trentaine contenant PARTITA, sur à peu près 50 sites. En particulier, PARTITA est utilisé chez DASSAULT dans le cadre de l'action GÉNIE. Dans cette action, DASSAULT fournira également des cas-tests pour l'utilisation du paralléliseur SPMD GASPARD.

References

[1] Inria. *"Centaur 2.0 Documentation"*, 1994.

[2] *"PADRE2 version 2α, user's manual"*, 1996.

[3] Simulog. *"Foresys 2.0 reference manual"*, 1998.

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

[5] Allen J., Kennedy K. *"Automatic translation of Fortran programs to vector form"*. *ACM Transactions on Programming Languages and Systems*, 9(4):491–542, 1987.

[6] Angeli L. *"Factorisation de sous-programmes"*. PhD thesis, Université de Nice Sophia-Antipolis, 1996.

[7] Applied Parallel Research. *"FORGExplorer"*, 1999.

[8] Attali I., Caromel D., Russo M. *"A formal and executable semantics for Java"*. In *Proceedings of Formal Underpinnings of Java, an OOPSLA'98 Workshop*, Vancouver, CA, 1998. Technical Report, Princeton University.

[9] Attali I., Pascual V., Roudet C. *"A language and an integrated environment for program transformations"*. Rapport de recherche 3313, Inria, 1997.

[10] Ayguadé E., Garcia J., Girones M., Grande M.L., Labarta J. *"DDT: a research tool for automatic data distribution in HPF"*. *Scientific Programming, special issue on HPF implementations*, 1996.

[11] Baker B. *"An algorithm for structuring flowgraphs"*. *Journal of the ACM*, 24(1):98–120, 1977.

[12] Bertot J., Bertot Y. *"CtCoq: A System Presentation"*. In M. McRobbie and J. Slaney, editors, *Automatic Deduction, CADE-13*, number 1104 in LNAI, pages 231–234. Springer Verlag, 1996.

[13] Bischof C., Carle A., Corliss G., Griewank A., Hovland P. *"ADIFOR: generating derivative codes from Fortran programs"*. *Scientific Programming*, 1(1):11–29, 1992.

[14] Bischof C., Carle A., Khademi P., Mauer A., Hovland P. *"ADIFOR2.0 User's Guide"*. Technical report, Argonne National Laboratory Technical Memorandum ANL/MCS-TM-192, and CRPC Technical Report CRPC-TR95516-S, 1998.

[15] Bischof C., Haghighat M.R. *"Hierarchical approaches to automatic differentiation"*. In M. Berz, C. Bischof, G. Corliss, A. Griewank, eds., SIAM, editor, *Computational Differentiation: Techniques, Applications and Tools*, pages 83–94, 1996.

[16] Blazy S. *"La spécialisation de programmes pour l'aide à la maintenance du logiciel"*. PhD thesis, Conservatoire National des Arts et Métiers, 1993.

[17] Bondorf A., Jones N.D., Mogensen T., Sestoft P. *"Binding time analysis and the taming of self-application"*. Technical report, DIKU, 1988.

[18] Borras P., Clement D., Despeyroux T., Incerpi J., Kahn G., Lang B., Pascual V. *"Centaur: the system". Proceedings of ACM SIGSOFT'88: Third Symposium on Software Development Environments, Boston*, November 1988. (Also as Inria Rapport de Recherche No 777).

[19] Boulet P. *"The Bouclettes loop parallelizer". Lecture Notes in Computer Sciences*, 1996. HPCN'96.

[20] Brezany P., Sipkova V., Chapman B., Greimel R. *"Automatic Parallelization of the AVL FIRE Benchmark for a Distributed-Memory System"*, 1995. ESPRIT Project PPPE report.

[21] Cacuci D.G. *"Sensitivity theory for nonlinear systems". J. Math. Phys.*, 22(12):2794–2812, 1981.

[22] Calland P.Y. *"Contributions aux techniques d'optimisation en compilation de programmes parallèles"*. PhD thesis, Ecole Normale Supérieure de Lyon, 1997.

[23] Campbell S.L., Hollenbeck R. *"Automatic differentiation and implicit differential equations"*. In M. Berz, C. Bischof, G. Corliss, A. Griewank, eds., SIAM, editor, *Computational Differentiation: Techniques, Applications and Tools*, pages 215–227, 1996.

[24] Cappelaere B., Elizondo D., Faure C. *"Using the Odyssée automatic differentiation tool for inverse modelling."*. In *Proceedings of European Geophysical Society, XXIV General Assembly*, 1999. to appear.

[25] Charpentier I. *"Génération de codes adjoints : Traitement de la trajectoire du modèle direct"*. Rapport de recherche 3405, Inria, 1998.

[26] Charpentier I., Ghemires M. *"Génération automatique de codes adjoints : Stratégies d'utilisation pour le logiciel odyssée, application au code météorologique meso-nh"*. Rapport de recherche 3251, Inria, 1997.

[27] Christianson B., Dixon L., Brown S. *"Sharing storage using dirty vectors"*. In M. Berz, C. Bischof, G. Corliss, A. Griewank, eds., SIAM, editor, *Computational Differentiation: Techniques, Applications and Tools*, pages 107–115, 1996.

[28] Clément D. *"A Distributed Architecture for Programming Environments"*. *Software Engineering Notes*, 15(5), 1990. Proceedings 4th Symposium on Software Development Environments.

[29] Clément D., Hascoët L. *"How to designate subtrees in CENTAUR"*. Technical report, INRIA, 1988. ESPRIT project GIPE, 3^{rd} annual report.

[30] Coleman T.F., Moré J.J. *"Estimation of sparse Jacobian matrices and graph coloring problems"*. *SIAM journal on numerical analysis*, 20:187–209, 1983.

[31] Cousot P. *"Abstract Interpretation"*. *ACM Computing Surveys*, 28(1):324–328, 1996.

[32] Curtis A.R., Powell M.J.D., Reid J.K. *"On the estimation of sparse Jacobian matrices"*. *J. Inst. Math. Appl.*, 13:117–119, 1974.

[33] Darte A. *"Techniques de parallélisation automatique de nids de boucles"*. PhD thesis, ENS Lyon, Université Lyon-1, 1993.

[34] Dutto P., Faure C. *"Extension of Odyssée to the MPI library : The direct mode"*. Rapport de recherche, INRIA, May 1999.

[35] Dutto P., Faure C., Fidanova S. *"Automatic differentiation and parallelism"*. In *Proceedings of Enumath 99, Finland*, 1999.

[36] Duval C., Eyrhard P., Faure C., Giilbert J.C. *"Application of the automatic differentiation tool odyssée to a system of thermohydraulic equations."*. In *Proceedings of ECCOMAS'96*, pages 795–802. Wiley, 1996.

[37] Eberhard P. *"Adjoint variable method for the sensitivity analysis of multibody systems interpreted as continuous, hybrid form of automatic differentiation"*. In M. Berz, C. Bischof, G. Corliss, A. Griewank, eds., SIAM, editor, *Computational Differentiation: Techniques, Applications and Tools*, pages 319–328, 1996.

[38] Faure C. *"Quelques aspects de la simplification en calcul formel"*. PhD thesis, Université de Nice, 1992.

[39] Faure C. *"Splitting of algebraic expressions for automatic differentiation.* In M. Berz, C. Bischof, G. Corliss, A. Griewank, eds., SIAM, editor, *Computational Differentiation: Techniques, Applications and Tools*, pages 117–127, 1996.

[40] Faure C., Duval C. *"Automatic differentiation for sensitivity analysis. A test case"*. In *Proceedings of Second International Symposium on Sensitivity Analysis of Model Output (SAMO'98)*, Venice (Italy), 1998. K. Chan, S. Tarantola and F. Campolongo editors, EUR report 17758-EN.

[41] Faure C., editor. *"Automatic differentiation for adjoint code generation."*. Rapport de recherche 3555, Inria, 1998.

[42] Faure C., Papegay Y. *"Odyssée user's guide. Version 1.7"*. Rapport technique 224, INRIA, 1998.

[43] Feautrier P. *"Semantical analysis and mathematical programming; application to parallelization and vectorization"*. In *Workshop on Parallel and Distributed Algorithms*, pages 309–320, Bonas, 1989. M. Cosnard, Y. Robert, P. Quinton, M. Raynal, editors, North Holland.

[44] Feautrier P. *"Dataflow analysis of array and scalar references"*. *Int. J. Parallel Programming*, 1(1):23–51, 1991.

[45] Ferrante J., Ottenstein K.J., Warren J.D. *"The Program Dependence Graph and its use in optimization"*. *ACM Transactions on Programming Languages and Systems*, 9(3):319–349, 1987.

[46] Frilley F. *"Différenciation d'ensembles structurés"*. PhD thesis, Université Paris VII, 1989.

[47] Futamura Y. *"Partial evaluation of computation process, an approach to a compiler-compiler"*. *Systems, Computers, Controls*, 2(5):45–50, 1971.

[48] Giboulot M.C., Thomasset F. Automatic parallelization of structured if statements without if conversion. In Marc Durand and Fadi El Dabaghi, editors, *High Performance Computing II (SHPC)*, pages 127–144, Amsterdam, 1991. Elsevier Science Publishers B.V., North-Holland. Version étendue : rapport de recherche RR–1408, INRIA, juin 1991.

[49] GIERING R. *"Tangent linear and Adjoint Model Compiler , Users manual 1.2"*, 1997.

[50] GIERING R., KAMINSKI T. *"Recipes for Adjoint Code Construction"*, 1998. in press ACM Trans. On Math. Software.

[51] GILBERT J.C. *"Automatic differentiation and iterative processes"*. *Optimization Methods and Software*, 1:13–21, 1992.

[52] GILBERT J.C., LEVEY G., MASSE J. *"La differentiation automatique de fonctions representees par des programmes"*. Rapport de recherche 1557, Inria, 1991.

[53] GRIEWANK A. *"Direct computation of Newton steps without accumulating Jacobians"*. In T.F. Coleman, Y. Li, eds., SIAM, editor, *Large-Scale Numerical Optimization*, pages 115–137, 1990.

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

[55] GRIEWANK A. *"Evaluating derivatives: principles and techniques of algorithmic differentiation"*. SIAM, 1999.

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

[57] GRIEWANK A., REESE S. *"On the calculation of Jacobian matrices by the Markowitz rule"*. In A. Griewank, G. Corliss, eds., SIAM, editor, *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, pages 126–135, 1991.

[58] GRIEWANK A., UTKE J., WALTHER A. *"Evaluating higher derivative tensors by forward propagation of univariate Taylor series"*. *To appear in Mathematics of Computation Journal*, 1998. Preprint IOKOMO-09-1997, Techn. Univ. Dresden.

[59] GRIMM J., POTTIER L., ROSTAING-SCHMIDT N. *"Optimal time and minimum space-time product for reversing a certain class of programs"*.

In M. Berz, C. Bischof, G. Corliss, A. Griewank, eds., SIAM, editor, *Computational Differentiation: Techniques, Applications and Tools*, pages 95–106, 1996. Rapport de Recherche INRIA 2794.

[60] HASCOËT L. *"A tactic-driven system for building proofs"*. 7^{eme} séminaire Programmation en Logique, Trégastel, France, 1988, aussi Rapport de Recherche INRIA n^{o} 770.

[61] HASCOËT L. *"Transformations automatiques de specifications semantiques: application: Un verificateur de types incremental"*. PhD thesis, Université de Nice Sophia-Antipolis, 1987.

[62] HASCOËT L. *"Partial Evaluation with Inference Rules"*. *New Generation Computing Journal*, 6(2 & 3):187–209, 1988.

[63] HASCOËT L. *"Specification of the PARTITA tool"*. Technical report, INRIA, 1994. EUREKA Report, Project 933 "EUROTOPS".

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

[65] HASCOËT L. *"Parallelization of Finite Element Codes with Automatic Placement of Communications"*. Rapport de recherche 3646, Inria, 1999.

[66] IRI M. *"History of automatic differentiation and rounding estimation"*. In A. Griewank, G. Corliss, eds., SIAM, editor, *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, pages 1–16, 1991.

[67] IRI M., TSUCHIYA T., HOSHI M. *"Automatic computation of partial derivatives and rounding error estimates with application to large-scale systems of nonlinear equations"*. *Journal of Computational and Applied Mathematics*, 24:365–392, 1988.

[68] KAHN G. *"Natural Semantics"*. *Lecture Notes in Computer Science*, 247, 1987. Proceedings of STACS 1987.

[69] KOELBEL C. *"Compiling Programs for Nonshared Memory Machines"*. PhD thesis, Purdue University, 1990.

[70] Kubota K. *"PADRE2: Fortran precompiler for automatic differentiation and estimates of rounding error"*. In M. Berz, C. Bischof, G. Corliss, A. Griewank, eds., SIAM, editor, *Computational Differentiation: Techniques, Applications and Tools*, pages 367–374, 1996.

[71] Kuck D.J. *"The structure of computers and computations"*, volume 1. Wiley, 1978.

[72] le Dimet F.X., Talagrand O. *"Variational algorithms for analysis and assimilation of meteorological observations: theoretical aspects"*. *Tellus*, 38A:97–110, 1986.

[73] McCormick G.P. *"Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem"*. *Mathematical Programming*, 26:153–171, 1983.

[74] Mohammadi B., Malé J.M., Rostaing-Schmidt N. *"Automatic differentiation in direct and reverse modes: application to optimum shapes design in fluid mechanics"*. In M. Berz, C. Bischof, G. Corliss, A. Griewank, eds., SIAM, editor, *Computational Differentiation: Techniques, Applications and Tools*, pages 309–318, 1996.

[75] Morgenstern J. *"How to compute fast a function and all its derivatives, a variation on the theorem of Baur-Strassen"*. *SIGACT News*, 16:60–62, 1985.

[76] Naumann U. *"Edge versus Vertex Elimination in Computational Graphs"*, 1998. Preprint TU Dresden.

[77] Naumann U. *"Optimizing the accumulation of jacobians by edge elimination in the computational graph"*. Rapport de recherche, INRIA, April 1999.

[78] Naumann U. *"SAVE: simulated annealing applied to the vertex elimination problem in computational graphs"*. Rapport de recherche, INRIA, April 1999.

[79] Naumann U. *"Efficient calculation of jacobians using dynamic programming"*. Rapport de recherche, INRIA, May 1999.

[80] Naumann U. *"Heuristics for efficiently calculating jacobians by edge elimination in computational graphs"*. Rapport de recherche, INRIA, May 1999.

[81] Newsam G.N., Ramsdell J.D. *"Estimation of sparse Jacobian matrices"*. *SIAM J. Alg. Disc. Meth.*, 4:404–417, 1983.

[82] OpenMP Architecture Review Board. *"OpenMP, Simple, Portable, Scalable SMP Programming"*, 1999.

[83] Pacific Sierra Research. *"VAST/Parallel, Fortran and C Automatic Parallelizing Preprocessors"*, 1999.

[84] Pingali K., Beck M., Johnson R., Moudgill M., Stodghill R. *"Dependence Flow Graphs: an algebraic approach to program dependencies"*. Pitman, 1991.

[85] Powell M.J.D., Tiont Ph.L. *"On the estimation of sparse Hessian matrices"*. *Journal of Numerical Analysis*, 16:1060–1074, 1979.

[86] Reps T., Horwitz S., Yang W. *"Detecting program components with equivalent behaviors"*. Technical Report 840, Computer Sciences Lab., University of Wisconsin, Madison, 1989.

[87] Reps T., Teitelbaum T. *"The Synthesizer Generator Reference Manual"*. Springer-Verlag, 1989.

[88] Rivière R. *"Un prototype d'environnement de génération de sous-programmes de résolution d'équations aux dérivées partielles"*. Technical report, Université de Nice Sophia-Antipolis, 1995. Mémoire de DEA "calcul et déduction".

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

[90] Rostaing N., Dalmas S., Galligo A. *"Automatic differentiation in Odyssée"*. *Tellus*, 45A(5):558–568, 1993.

[91] Shah P. *"Application of adjoint equations to estimation of parameters in distributed dynamic systems"*. In A. Griewank, G. Corliss, eds., SIAM, editor, *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, pages 181–190, 1991.

[92] SOGNO J.C. *"Analysis of multidimensional loops with non-uniform dependences"*. In *Proceedings of Int. Conf. on Advances in Parallel and Distributed Computing, APDC'97*, 1997.

[93] TADJOUDDINE M., EYSSETTE F., FAURE C. *"Sparse Jacobian Computation in Automatic Differentiation by Static Program Analysis"*. *Lecture Notes in Computer Science*, 1503:311–326, 1998.

[94] TALAGRAND O. *"The use of adjoint equations in numerical modelling of the atmospheric circulation"*. In A. Griewank, G. Corliss, eds., SIAM, editor, *Automatic Differentiation of Algorithms: Theory, Implementation and Application*, pages 169–180, 1991.

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