INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet A3

Analyse Avancée Appliquée à l'optimisation de code

Rocquencourt

THÈME 1A

Table des matières

1 Composition de l'équipe

Responsable scientifique

Christine Eisenbeis [CR Inria]

Assistante de projet

Nathalie Gaudechoux [SAR Inria]

Personnel Inria

Albert Cohen [CR]

Paul Feautrier [Professeur de l'Université de Versailles–Saint-Quentin, en délégation depuis le 1er octobre 2000]

François Thomasset [DR]

Personnel PRISM, Université de Versailles–Saint-Quentin

Denis Barthou [Maître de conférences, Université de Versailles-Saint-Quentin]

Collaborateur extérieur

Daniela Genius [Bourse Marie Curie, Philips Research France]

Doctorants

Christophe Alias [bourse MENRT, Université de Versailles-Saint-Quentin, depuis le 1er septembre 2001]

Pierre Amiranoff [professeur certifié de mathématiques, détaché comme 1/2 ATER IIE (Evry)-CNAM, puis 1/2 ATER CNAM-Paris depuis le 1er septembre 2001]

Cédric Bastoul [bourse MENRT, Université de Paris VI]

Ivan Djelic [bourse MENRT, Université de Paris VI, puis 1/2 ATER, Université de Versailles St-Quentin, à partir du 1er septembre 2001]

Andry Randrianatoavina [bourse MENRT, Université de Strasbourg, jusqu'au 30 septembre 2001]

Sid Ahmed Ali Touati [bourse INRIA]

Stagiaires

Aloke Bajpai [Indian Institute of Technology, Delhi, Inde, du 1er juin au 20 juillet 2001]

Nikhil Bansal [Indian Institute of Technology, Delhi, Inde, du 1er mai au 30 juillet 2001]

Patrick Carribault [Maîtrise, Université de Versailles, du 1er juillet au 31 août 2001]

Nourredine Djaballah [Stage de DEA, Université d'Orléans, du 1er avril au 31 août 2001]

Abdesselem Kortebi [Stage commun avec Philips Research France, Université de Paris VI, du 1er juin au 13 septembre 2001]

Tarek Sahli [Stage de DEA, Université d'Orléans, du 1er avril au 31 août 2001]

2 Présentation et objectifs généraux

Avant-projet depuis 1996, le projet A3 a été créé en décembre 1998. A3 est un projet commun entre l'INRIA et le laboratoire PRISM de l'université de Versailles-Saint-Quentin, agréé de plus par le CNRS depuis le 9 juillet 1999. Les recherches portent sur l'analyse de programmes, avec ses applications en optimisation de la performance des codes sur les nouvelles générations d'ordinateurs, en particulier l'optimisation de la gestion des hiérarchies mémoire

et du parallélisme d'instructions. A3 élabore des méthodes et des outils destinés à être utilisés par le compilateur ou l'utilisateur pour analyser et transformer les codes, afin qu'ils exploitent au mieux les spécificités architecturales de la machine.

Le projet A3 a pour objectifs :
- de développer de nouvelles méthodes d'analyse de flots de données dans les programmes,
- d'appliquer les méthodes traditionnelles d'analyse statique à l'optimisation de code,
- de prendre en compte des caractéristiques architecturales dans la phase d'analyse de code,
- de développer de nouvelles méthodes d'optimisation de code,
- de développer des méthodes et outils d'analyse dynamique de code et des méthodes d'optimisation prenant en compte les résultats de ces analyses.

Du côté des **applications**, A3 vise particulièrement :
- l'optimisation des programmes, dits de calcul intensif, sur les processeurs à haute performance présents dans les PC et stations de travail ;
- l'optimisation de codes sur les processeurs spécialisés et/ou embarqués ;
- la parallélisation des programmes sur les serveurs de calcul (stations de travail à petit nombre de processeurs).

3 Fondements scientifiques

Logiciel ou matériel ? La programmation des ordinateurs est un éternel compromis entre les deux. Processeurs spécialisés à un extrême, microprocesseurs généralistes de l'autre, ce problème est amplifié dans la recherche de la performance. En effet, les architectures de processeurs à haute performance sont en constante évolution et leur programmation efficace requiert une expertise de plus en plus pointue. Alors qu'il suffisait de « vectoriser » ou de « paralléliser » son programme – notions *de haut niveau*, gérables dans le programme source – sur les supercalculateurs du début des années 80, il faut aujourd'hui tenir compte de la hiérarchie mémoire et du parallélisme d'instructions – notions plus fines typiquement gérées dans le code machine.

La phase d'**analyse sémantique du programme**, de ses schémas d'accès aux données, ainsi que de son comportement prévisible à l'exécution est un préalable à toute optimisation. Dans les compilateurs classiques, l'analyse de code s'appuie sur des bases théoriques solides de sémantique de programmes et s'applique à tout type de code, mais les informations qu'elle calcule sont peu précises, en particulier en ce qui concerne les données structurées. Au contraire, les paralléliseurs automatiques effectuent une analyse particulièrement fine des accès aux tableaux, mais l'analyse est restreinte aux codes à contrôle régulier (boucles) et aux accès réguliers à la mémoire.

De même, les optimisations réalisées par les compilateurs classiques concernent principalement la réduction du nombre de calculs à exécuter – par exemple, l'étude des *invariants de boucle* évite de répéter un même calcul à chaque itération. Ces méthodes s'appliquent à tout type de programme et se basent sur la sémantique de celui-ci. En revanche, les méthodes d'optimisation pour les architectures à haute performance sont basées sur une transformation de l'ordre d'exécution des instructions. Elles sont très efficaces dans les cas restreints de code linéaire ou boucle sans branchements – pour le parallélisme d'instructions – et des accès régu-

liers aux tableaux – pour la gestion de la hiérarchie mémoire ; leur extension à des programmes quelconques reste un problème ouvert.

Ainsi, aussi bien en analyse qu'en optimisation de code, deux grandes classes de méthodes se dessinent. La première est généraliste, mais ne prend pas en compte les spécificités architecturales. La seconde est spécialisée, mais est restreinte à certaines constructions de programmes. Comment conjuguer généralité et efficacité ? C'est autour de ce problème que s'articule le projet A3.

Nous développons ci-après les fondements de deux axes du projet, l'analyse statique de code et le parallélisme d'instructions. Les deux autres thèmes, la gestion de la mémoire et les problèmes d'allocation de registres, sont directement explicités dans la partie « Résultats ».

3.1 Analyse de code

Les analyses statiques de code sont nombreuses. En nous restreignant aux analyses portant sur les accès à la mémoire dans les programmes impératifs, citons simplement l'analyse de dépendance et l'analyse de flot de données.

Les analyses de dépendance déterminent les couples d'opérations en conflit mémoire (les analyses d'alias sont conceptuellement identiques mais retournent les couples d'accès mémoire en conflit). L'analyse de flot de données, quant à elle, ne retient de ces couples d'opérations que celui livrant la dernière écriture précédant une lecture donnée. Cette dernière écriture, appelée la *source*, produit la *valeur* lue, c'est donc bien elle qui nous informe sur le flot des données. L'analyse de flot de données est utilisable aussi bien pour la mise au point (recherche des variables non initialisées) que pour l'analyse de localité ou la parallélisation automatique.

Les analyses de flot de données s'appuient sur une des deux principales classes techniques que nous appellerons respectivement itératives[1] et géométriques. Les analyses itératives, plus classiques et dans la ligne des travaux de Floyd [Flo67], cherchent à associer une propriété à chaque point du programme [KU76]. Par exemple, on cherchera à prouver que juste avant chaque exécution d'une certaine instruction, une certaine variable est positive, ou bien a une valeur fixée (propagation des constantes), ou bien encore que plusieurs variables ont des valeurs qui vérifient une certaine relation [Cou81]. Les assertions que l'on manipule sont éléments d'un treillis ordonné par la relation « contenir plus d'information que ... ». On cherche à montrer que les propriétés cherchées satisfont à une équation de point fixe. L'existence de la solution est assurée si les opérateurs qui apparaissent dans l'équation sont monotones. La solution peut être trouvée par itération si la hauteur du treillis est finie ou si l'on dispose d'un opérateur d'élargissement [CC77].

[1] Nous appelons ces méthodes ainsi bien qu'elles soient en fait caractérisées par le système d'équations de flots qu'elles posent, système qui peut être résolu dans des cas simples par des méthodes directes.

[Flo67] R. W. Floyd, « Assigning Meaning to Programs », *in : Proc. of the Symp. in Applied Mathematics, Vol. 19*, J.T.Schwartz (éditeur), AMS, p. 19–32, Providence, 1967.

[KU76] J. Kam, J. Ullmann, « Global Data Flow Analysis and Iterative Algorithms », *Journal of the ACM 23*, 1, janvier 1976, p. 158–171.

[Cou81] P. Cousot, *Program Flow analysis: theory and applications*, Prentice-Hall, 1981, ch. Semantic foundations of programs analysis, p. 303–342.

[CC77] P. Cousot, R. Cousot, « Abstract Interpretation: A Unified Lattice Model for Static Analysis

L'analyse géométrique [6] du flot des données dans les tableaux procède d'une manière toute différente. Le but est de relier chaque valeur lue à sa *source*, c'est-à-dire à l'opération qui l'a écrite. Les ensembles d'opérations sont représentées par des polyèdres et le calcul de la source se ramène à des opérations d'union, d'intersection et de recherche de maximum sur ces polyèdres. Si le programme est régulier et à contrôle statique, la source est unique.

La compréhension des liens entre ces deux méthodes constitue un sujet de recherche en soi, toujours ouvert à l'heure actuelle. Nos recherches sont des premiers pas vers une éventuelle unification.

3.2 Parallélisme d'instructions

Les microprocesseurs modernes disposent en général d'un petit nombre d'unités fonctionnelles indépendantes et peuvent exécuter plusieurs instructions simultanément si les dépendances de données s'y prêtent et si les ressources nécessaires sont disponibles. Le choix des instructions à lancer peut être laissé au compilateur (architectures VLIW), ou au matériel (architectures superscalaires). Dans ce dernier cas, comme le matériel ne prend en compte qu'un nombre limité d'instructions candidates, le compilateur peut encore agir sur les performances du programme en réordonnant les instructions.

Un premier type d'optimisation (*local scheduling, trace scheduling, percolation scheduling*, compaction) s'applique au code *linéaire*, c'est-à-dire sans branchement (bloc de base, trace de programme) et nécessite une analyse fine du flot des données dans le programme. Dans le cas des boucles, le but de l'exercice est de construire un pipeline logiciel, dans lequel plusieurs itérations de la même boucle sont actives simultanément, de façon à saturer les ressources disponibles. La méthode d'ordonnancement cyclique (*modulo scheduling*), due à Rau [RG81], construit une table de réservation des ressources disponibles en prenant en compte le caractère périodique du déroulement du programme. De notre côté, nous avons développé dans le passé [10] l'algorithme DESP (Decomposed Software Pipelining) qui réalise le pipeline logiciel en se ramenant au réordonnancement d'un code linéaire.

Les points non résolus de manière satisfaisante sont la prise en compte des branchements, les problèmes d'allocation dans les registres, l'interaction avec le problème de la gestion de la hiérarchie mémoire, ainsi que l'interaction avec les méthodes de parallélisation automatique. Lorsque le temps de compilation n'est pas un obstacle, on peut envisager des méthodes exactes d'optimisation par programmation linéaire [Han94,Fea94,GAG94], voir aussi [9]. Le problème se

of Programs by Construction or Approximation of Fixpoints », *in : 4th POPL, Los Angeles, CA*, p. 238–252, janvier 1977.

[RG81] B. R. Rau, C. D. Glaeser, « Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing », *in : Proceedings of the* 14^{th} *Conference on Microprogramming and Microarchitecture*, p. 183–198, octobre 1981.

[Han94] C. Hanen, « Study of a NP-hard cyclic scheduling problem: the recurrent job-shop », *European Journal of Operational Research 72*, January 1994, p. 82–101.

[Fea94] P. Feautrier, « Fine-Grain Scheduling under Resource Constraints », *in : 7th Workshop on Language and Compilers for Parallel Computing*, Springer-Verlag, LNCS 892, p. 1–15, août 1994.

[GAG94] R. Govindarajan, E. Altman, G. Gao, « A framework for Ressource-Constrained Rate-Optimal Software Pipelining », *in : Conference on Vector and Parallel Processing(CONPAR-94 VAPP VI)*, Linz, Austria, september 1994.

réduit alors à un problème de modélisation des contraintes.

4 Domaines d'applications

Le domaine d'application de A3 est essentiellement l'optimisation des codes dans les architectures à haute performance. Dans ces architectures, nous incluons les microprocesseurs généralistes, les processeurs embarqués spécialisés ou les DSP, mais aussi les serveurs de calcul à petit nombre de processeurs, ou encore les supercalculateurs présentant un modèle de programmation à mémoire partagée.

Du côté des programmes, les applications visées sont celles qui sont critiques en performance. Entrent dans ce cadre les programmes de calcul scientifique (projet MHAOTEU), ou les applications de type multimédia (projet OCEANS).

Les méthodes et algorithmes développés dans A3 peuvent s'appliquer à tout niveau de la chaîne de programmation :

- environnement de programmation (ré-ingéniérie de code, outils interactifs d'optimisation avec profiling et boîte à outils de transformations). Le projet ESPRIT MHAOTEU (section 8.2.1) vise ce type d'applications pour l'optimisation de la hiérarchie mémoire.
- pré-processeur d'optimisation source à source : c'est le cas de PAF (Paralléliseur Automatique de FORTRAN) ou de TOPS (pipeline logiciel source à source, section 5.3).
- compilateur ;
- post-processeur d'optimisation assembleur vers assembleur, comme dans la plate-forme SALTO du projet CAPS de l'IRISA, auxquels nos logiciels PiLo et LoRA sont intégrés ;
- de plus, les travaux de A3 permettent de caractériser la difficulté d'utilisation de chaque dispositif des processeurs spécialisés, et donc d'influer sur leur architecture.

Le projet ESPRIT OCEANS, achevé fin 1999, s'articulait justement autour de l'interaction entre les 2 phases de pré-processing et de post-processing, pour la génération de code performant pour les architectures VLIW.

5 Logiciels

5.1 PiLo : pipeline logiciel

PiLo est un package de pipeline logiciel interfaçable, développé par Antoine Sawaya dans sa thèse [9]. PiLo est basé sur une modélisation de la boucle (de type **FOR**) à optimiser, ainsi que des contraintes architecturales du processeur. La boucle est donnée sous la forme de son graphe de dépendances de données, spécifiant les latences d'exécution ainsi que les distances de dépendance. On peut aussi préciser pour chaque variable portée par une dépendance si le renommage est autorisé ou non. Les contraintes architecturales sont spécifiées sous la forme de tables de réservation, nombre d'unités fonctionnelles et nombre de registres disponibles. En sortie, PiLo donne un ordonnancement de pipeline logiciel, avec prologue, état permanent et épilogue.

PiLo est basé sur la méthode *DESP* (Decomposed Software Pipelining [10]), améliorée dans [9]. Il est utilisé dans l'environnement Sage++ (transformation source à source) ainsi

que dans l'environnement SALTO (projet ESPRIT OCEANS). La nouvelle version de SALTO développée dans le projet CAPS de l'IRISA s'appelle ALISEE. Nikhil Bansal [28] a travaillé durant son stage sur la connection de PiLo à ALISEE. Pour cela, il a dû définir une interface XML pour PiLo.

5.2 LoRA : allocation de registres dans les boucles

LoRA [5] est un package d'allocation de registres dans les boucles, développé par Sylvain Lelait dans sa thèse [7]. LoRA est basé sur le *meeting graph* (voir section 5.2). Le but est de trouver un compromis entre nombre de registres utilisés et déroulage de la boucle nécessaire à l'allocation.

LoRA prend en entrée une famille d'intervalles circulaires spécifiée par la taille du cercle, les points extrêmaux de chaque intervalle ainsi qu'un nombre de registres disponibles. On peut aussi donner des types différents pour chaque intervalle et un nombre de registres par type. LoRA calcule un degré de déroulage et l'allocation pour chaque instance d'intervalle dans la boucle déroulée. Selon les options, on peut spécifier la recherche du degré minimal de déroulage ou du nombre minimal de registres, et différentes heuristiques.

LoRA est interfacé avec PiLo (voir ci-dessus), et a aussi été intégré dans l'environnement MOST (Modulo Scheduling Testbed), développé à l'Université de McGill (Montreal).

5.3 TOPS : pipeline logiciel source à source

TOPS est un outil permettant de pipeliner les boucles dans les programmes source (FORTRAN). L'utilisateur ou le compilateur désigne les boucles à pipeliner en insérant des directives spéciales, et peut aussi spécifier, après analyse ou par expérience personnelle, quelles données devraient être chargées en avance, pour éviter de bloquer le processeur en cas de défaut de cache. TOPS a été développé par Min Dai dans sa thèse [Dai00]. Il utilise l'environnement Sage++ de manipulation de programmes.

5.4 SAREQ : test d'équivalence de systèmes d'équations de récurrence

Le logiciel SAREQ a été développé par Xavier Redon (LIFL, Université de Lille I) dans le cadre de sa collaboration avec Denis Barthou et Paul Feautrier sur la reconnaissance d'algorithmes (voir Section 6.1.5). Il permet de tester l'équivalence de deux systèmes d'équations de récurrence affines. Il est écrit pour l'essentiel en Ocaml et utilise les bibliothèques polyédriques « Polylib » et « Omega ». Le logiciel est accessible directement sur le Web à l'URL `sarequ.eudil.fr` au travers d'un interface php. Cet interface permet l'entrée des SAREs à comparer dans une notation analogue à ALPHA, comporte une banque d'exemples, et renvoie un compte-rendu détaillé après chaque test.

[Dai00] M. Dai, *Transformation de code de haut niveau*, thèse de doctorat, Université Versailles–Saint-Quentin-en-Yvelines, 9 mai 2000.

6 Résultats nouveaux

6.1 Analyse sémantique des programmes

6.1.1 Analyse de dépendances en présence de variables inductives généralisées

Participant : Albert Cohen.

En collaboration avec Peng Wu, David Padua et Jay Hoeflinger (Université de l'Illinois à Urbana-Champaign, USA).

On ne compte plus les techniques d'analyse et les outils de parallélisation automatique pour les applications de calcul intensif en Fortran. Pourtant, la détection du parallélisme reste très souvent en deçà des espérances lors de la confrontation de ces outils avec des codes réels. Les deux raisons invoquées sont fortement interdépendantes et bien connues : détecter et exploiter le parallélisme est difficile et cher (NP-complétude de la plupart des problèmes d'analyse et d'optimisation agressifs), on se contente donc souvent d'approximations et de simplifications excessives du modèle.

Ainsi, de nombreux nids de boucles des benchmarks Perfect Club comportent des boucles parallèles que les compilateurs existants ne détectent pas, à cause des irrégularités dans l'évolution des variables inductives (scalaires liés aux compteurs de boucles à travers des équations récurrentes). Nous avons remarqué que ces irrégularités sont le plus souvent peu significatives en termes de dépendances entre accès mémoire, et que dans ces cas précis, des comparaisons locales portant sur la monotonie des parcours de tableaux permettent de conclure à l'absence de dépendances. En étendant le concept d'évolution monotone [GSW95] à des intervalles locaux de l'exécution, nous avons défini un algorithme incrémental mais non itératif avec une complexité étonnamment basse pour un test de dépendance en présence de variables inductives. Cet algorithme permet également d'accélérer la détection de parallélisme dans des codes où seules des techniques coûteuses étaient applicables. Un article est paru dans une conférence [18], et deux extensions sont parues dans les actes de workshops [11], [19].

6.1.2 Analyse des références en Java

Participants : Albert Cohen, Paul Feautrier.

En collaboration avec Peng Wu et David Padua (Université de l'Illinois à Urbana-Champaign, USA).

De nombreux chercheurs considérent Java comme un langage de programmation universel, et pensent qu'il remplacera de plus en plus souvent Fortran, C++, et, dans une moindre mesure, C. Les raisons de cet engouement sont la portabilité de Java, son caractère distribué et sa sécurité d'emploi. L'obstacle principal à l'utilisation généralisée de Java vient de son inefficacité. Les faibles performances sont dues entre autres à l'utilisation d'une machine virtuelle, à la présence de nombreux tests de validité à l'exécution, et à la quasi-impossibilité d'exploiter les optimisations classiques par suite de l'utilisation intensive des références. Il s'ensuit que toute

[GSW95] M. P. Gerlek, E. Stoltz, M. Wolfe, « Beyond induction variables: detecting and classifying sequences using a demand-driven SSA form », *ACM Trans. on Programming Languages and Systems 17*, 1, janvier 1995, p. 85–122.

tentative d'optimisation de Java doit passer par une phase d'analyse des références, et que meilleurs seront les résultats de cette analyse, plus il sera possible d'optimiser.

Sujet principal de notre collaboration avec Peng Wu et David Padua (de l'Université de l'Illinois à Urbana-Champaign) depuis 1999, l'étude du problème s'est focalisée sur les structures de données de type *conteneurs* (des listes, vecteurs, arbres, graphes acycliques, tables de hachage...). On exploite les propriétés sémantiques de ces objets définies dans des bibliothèques standard afin d'étendre une analyse de pointeurs/références traditionnelle [SRW98]. Au lieu de nommer directement des positions à l'intérieur de ces conteneurs, il est plus simple et plus général de procéder à une analyse spécialisée fondée sur la notion de « peigne » : un conteneur dont chaque élément donne accès à une structure de données isolée. Cette analyse permet de paralléliser des parcours de conteneurs, d'optimiser les chargements en mémoire, de supprimer des redondances, etc. Une preuve de correction a été réalisée dans un formalisme d'interprétation abstraite, mais de nombreuses questions restent ouvertes (notions d'optimalité, comparaison plus étendue avec le formalisme de [SRW99], optimisation/simplification du treillis, implémentation efficace, extension inter-procédurale). Un rapport de recherche est paru ([22]).

6.1.3 Analyse de code prédicaté

Participants : Ivan Djelic, Albert Cohen.

L'élimination des redondances partielles, la propagation de constantes, l'allocation de registres et l'ordonnancement des instructions constituent autant d'optimisations cruciales pour les architectures récentes. Ces techniques sont bien maîtrisées, mais un nouveau mécanisme architectural — présent notamment sur l'IA 64 (Itanium) — remet en cause les conditions d'application et les performances des algorithmes existants : il s'agit de la « prédication », c'est-à-dire l'exécution conditionnelle des instructions. Dans le cadre de l'élimination de redondances partielles, un nouvel article sur le sujet paraîtra en 2002 dans un journal [32]. Des expérimentations ont été menées dans l'environnement Trimaran (compilateur IMPACT et simulation d'architecture EPIC) ; une preuve de correction et d'optimalité a été obtenue, à l'aide d'un nouveau treillis représentant les interactions entre prédicats avec un maximum d'expressivité et un minimum de complexité algorithmique.

Les travaux se poursuivent vers l'extension de cette approche à d'autres optimisations de code, vers une formalisation et une preuve en interprétation abstraite, et vers des expérimentations systématiques sur des benchmarks.

6.1.4 Analyse de programmes utilisant des pointeurs

Participants : Pierre Amiranoff, Albert Cohen, Paul Feautrier.

Les programmes utilisant des pointeurs sont difficiles à optimiser, car il est difficile de savoir

[SRW98] M. Sagiv, T. Reps, R. Wilhelm, « Solving Shape-Analysis Problems in Languages with Destructive Updating », *ACM Trans. on Programming Languages and Systems 20*, 1, janvier 1998, p. 1–50.

[SRW99] S. Sagiv, T. W. Reps, R. Wilhelm, « Parametric Shape Analysis via 3-Valued Logic », *in: 26th Annual ACM Symp. on Principles of Programming Languages (PoPL'99)*, p. 105–118, San Antonio, Texas, USA, janvier 1999.

quel est l'objet pointé à la compilation. Les pointeurs permettent de construire des structures de données complexes (par exemple des arbres) qui n'apparaissent pas explicitement dans le texte source. Enfin, de même que l'usage des instructions `goto` encourageait l'écriture de programmes à structure de contrôle anarchique, l'usage de pointeurs encourage la construction de structures de données anarchiques.

Une façon d'éviter cet obstacle est de promouvoir l'utilisation de programmes à structures de données régulières, en un sens à définir. Le cas le plus simple est celui des tableaux, qui a été l'objet de multiples recherches ces dernières années. Viennent ensuite les arbres, qui sont à la base de multiples algorithmes non numériques. De même que la boucle est la structure de contrôle privilégiée pour la manipulation des tableaux, la récursion est la structure de contrôle adaptée à la manipulation des arbres. D'où notre intérêt pour les programmes récursifs sur les arbres, qui sont l'objet entre autres d'une part substantielle de la thèse d'Albert Cohen.

Le synopsis de l'étude est le suivant :

- on commence par identifier les pointeurs qui se comportent comme une adresse dans un arbre : on les reconnaît à ce que la seule opération permise est la déréférenciation (*pointer chasing*). Ces pointeurs sont les analogues des variables inductives en analyse de tableaux ; nous les appelons des *variables inductives généralisées*.
- on calcule ensuite la valeur d'une telle variable en fonction de l'instant d'exécution du programme. Un instant d'exécution est un mot (le mot de contrôle) qui représente, en gros, la position d'un appel de fonction dans l'arbre (dynamique) des appels. La relation entre un instant d'exécution et la valeur d'un pointeur est représentée par un transducteur rationnel, le *transducteur d'adresses*.
- enfin, le calcul des dépendances se ramène à des problèmes d'intersection de relations rationnelles. Il s'agit d'un problème indécidable bien connu, pour lequel Paul Feautrier et Albert Cohen ont défini plusieurs semi-algorithmes.

Pierre Amiranoff [20] a repris l'étude dans le cadre de sa thèse. Il donné une définition formelle du mot de contrôle en s'appuyant sur la théorie des traces. Il a caractérisé de façon précise les variables inductives généralisées, et il a donné un algorithme de calcul du transducteur d'adresses plus simple que celui proposé dans la thèse d'Albert Cohen [1].

Cette étude peut se poursuivre dans plusieurs directions :

- les transducteurs d'adresses actuels représentent des fonctions. Mais un transducteur rationnel peut sans changement de formalisme représenter une relation. Cette modification simple permettrait de traiter de nouvelles structures de contrôle (par exemple les conditionnelles) et certains des pointeurs qui ne sont pas des variables inductives suivant la définition actuelle.
- la structure d'arbre est la plus simple des structures récursives, mais il en existe bien d'autres, comme les listes doublement chaînées ou les arbres où chaque sommet porte un pointeur vers son père. Comme plusieurs chercheurs (L. Hendren, J-L. Giavitto) l'ont déja remarqué, ces structures sont des monoïdes finiment présentés (i.e. des monoïdes spécifiés par un ensemble de générateurs et des relations entre générateurs). Nous proposons d'appeler *structure de données régulière* une structure de données dont l'ensemble des positions est (une partie d') un monoïde finiment présenté. Le problème est alors double : pour une structure concrète donnée, identifier son type, puis développer un test de dépendance adapté à ce type.

– enfin, tout reste à faire en ce qui concerne les applications de cette étude. Presque toutes les optimisations connues passent de façon plus ou moins explicite par une phase d'analyse de dépendances. Si Albert Cohen a obtenu quelques résultats en matière de parallélisation, l'application de nos résultats à des optimisations plus classiques est un domaine entièrement ouvert.

6.1.5 Reconnaissance d'Algorithmes

Participants : Denis Barthou, Paul Feautrier.

En collaboration avec Xavier Redon (LIFL, Lille I).

Reconnaître un algorithme, c'est décider qu'un programme dont on donne le texte en langage de haut (ou bas niveau) est une implémentation d'un algorithme dont on a la spécification en langage de très haut niveau. Le problème est relié a celui de l'équivalence de deux machines de Turing ; il est bien connu qu'il est indécidable, et qu'il le reste même si l'on impose des restrictions fortes aux programmes traités (par exemple, si l'on garantit leur terminaison). Le problème est cependant d'un grand intérêt pour l'optimisation, la parallélisation et la vérification de programmes. Certains compilateurs possèdent donc des modules de reconnaissance plus ou moins élaborés, par exemple pour reconnaître les opérations de réduction. Ces modules procèdent par reconnaissance de forme après une étape de normalisation.

Nous nous intéressons à ce problème depuis la thèse de X. Redon qui portait sur la reconnaissance des réductions. L'originalité de ce travail était la méthode de normalisation, qui passait par la mise en forme à assignation unique du programme source. Cette forme permet de s'abstraire à la fois de la structure des données (suppression de la distinction tableau/scalaire) et du contrôle. Nous sommes passé de là au problème de l'équivalence de deux SAREs, toujours indécidable mais pour lequel nous avons pu définir un semi-algorithme, sous l'hypothèse que les opérateurs de calculs n'ont pas de propriétés spéciales. Nous avons prouvé la complétude de cet algorithme [21] et implémenté un prototype accessible à l'URL `http://sareq.eudil.fr`. Cette implémentation dépend de la possibilité de calculer la fermeture transitive d'une relation de Pressburger, calcul qui n'est possible que dans certains cas particuliers au demeurant très fréquents.

Nous comptons poursuivre ce travail dans plusieurs directions :
– amélioration du calcul de la fermeture transitive et optimisation du prototype.
– introduction des propriétés sémantiques de opérateurs.
– construction d'un environnement d'analyse et d'optimisation de programmes basé sur la reconnaissance d'algorithmes. Ce dernier travail est le sujet de la thèse en cours de Christophe Alias.

6.2 Analyse des comportements des programmes et modélisation des architectures

6.2.1 Modélisation et vérification d'applications de traitement vidéo

Participants : Albert Cohen, Daniela Genius, Paul Feautrier, Abdesselem Kortebi.

Dans le cadre de notre projet avec Philips Research France, et après avoir identifié un

domaine d'applications spécifiques (flux vidéo, 3D, temps réel, régularité et contrôle statique), nous avons introduit un nouveau modèle appelé *réseaux hiérarchiques de processus* ou *hierarchical process networks* (HPN). Ce modèle permet de traiter un ensemble restreint d'applications de traitement du signal en temps réel, mais il décrit très précisément l'exécution et les ressources (unités fonctionnelles, mémoire, bande passante) nécessaires. Les deux caractéristiques principales des HPN sont les suivantes : d'une part une description explicite de la structure hiérarchique des données (pixels, tuiles, lignes, etc.) et des événements associés, d'autre part un modèle d'activation périodique « relâché » (avec rafales et variations) mais avec des bornes maximales garanties.

Un prototype a été réalisé lors du stage d'Abdesselem Kortebi. La validation pratique du modèle par rapport aux algorithmes de traitement vidéo est en cours. Les recherches s'orientent vers une extension du modèle théorique et vers les interactions avec l'ordonnancement, la description de la machine, et l'allocation de ressources (pipeline logiciel hiérarchique).

6.2.2 Fenêtres de références généralisées - approche dynamique

Participants : Aloke Bajpai, Christine Eisenbeis, Andry Randrianatoavina.

Pour estimer la localité dans les programmes, nous utilisons la notion de fenêtres de référence généralisées aux caches associatifs. Cette approche est basée, non sur l'estimation directe du nombre de défauts de cache, mais sur le nombre de variables en vie dans chaque *cache set*.

Statiquement, ce nombre s'évalue comme le nombre de points entiers dans un polyèdre paramétré, ou plus exactement dans la projection d'un tel polyèdre sur un sous-espace. Cette approche est toujours en cours d'étude. Basée sur les polynômes d'Ehrhart, dont une généralisation et une implémentation ont été développées par Philippe Clauss à Strasbourg, elle bute sur les problèmes de projection.

C'est pourquoi nous développons aussi un outil de visualisation au vol de la taille des fenêtres de référence. Aloke Bajpai est revenu cette année travailler sur ce logiciel, DGVT (Dynamic General reference window Visualisation Tool), qu'il avait commencé à développer chez nous en 2001. Ce logiciel peut être utilisé comme un *debugger* de performances.

Nous étudions ces problèmes en partenariat avec l'Université de Strasbourg.

6.3 Optimisation des programmes - mémoire

6.3.1 Amélioration de la localité des programmes réguliers

Participants : Cédric Bastoul, Paul Feautrier.

Les concepteurs des caches se sont basés essentiellement sur des arguments probabilistes : chaque programme adresse la mémoire centrale au hasard (ce qui ne veut pas dire de manière uniforme). Si ce modèle convient bien pour les programmes irréguliers usuels, il tombe en défaut pour les programmes réguliers de type calcul scientifique ou traitement du signal. De plus les applications embarquées demandent que les temps d'exécution soient prédictibles, et donc que les défauts de caches soient maîtrisés.

Pour obtenir ce résultat, nous avons proposé de « court-circuiter » le mécanisme de remplacement, dont le fonctionnement est très difficile à prévoir. Le programme est divisé en

morceaux. Au début de chaque morceau, le cache est vide. La taille du morceau est ajustée de façon que ses données tiennent juste dans le cache. Le mécanisme de remplacement n'intervient donc pas. A la fin du morceau, le cache est vidé et on passe au morceau suivant.

Dans ce modèle, il est possible de donner une estimation asymptotique du trafic et d'utiliser cette information pour calculer un découpage optimum. Le découpage peut donner lieu, le cas échéant, à des restructurations du programme source. Dans le cadre de son travail de thèse, Cédric Bastoul a défini les algorithmes de découpage pour les cas les plus importants : auto-réutilisation simple et multiple. Il a réalisé un prototype. A partir du découpage obtenu, il faut reconstruire le programme objet. Il termine actuellement l'implémentation de cette phase, ce qui lui permettra de mener une campagne de mesures intensives, et d'évaluer l'importance de la réutilisation de groupe.

Dans la version actuelle, seule la localité temporelle est prise en compte, et aucun tuilage n'est effectué. L'exploitation de la localité spatiale et le tuilage feront l'objet de la deuxième année de thèse de Cédric.

A plus long terme, il est souhaitable de transposer ces recherches aux mémoires locales qui remplacent souvent les caches sur les processeurs embarqués. Les mémoire locales ne disposent ni de mécanisme de remplacement, ni de système d'adressage associatif. Il faut donc que le compilateur prenne entièrement en charge la gestion de l'espace mémoire. Un constructeur de systèmes embarqués a déjà manifesté de l'intérêt pour cette extension.

6.3.2 Minimisation des défauts de cache par programme linéaire

Participants : Albert Cohen, Christine Eisenbeis, Tarek Sahli.

L'approche généralement adoptée dans l'optimisation de la localité des programmes est d'essayer une à une les transformations possibles, et d'évaluer d'une manière ou d'une autre leur effet. Nous avons choisi de poser le problème globalement : étant donnée la liste des accès à la mémoire d'un code, reliés par des contraintes de précédence dues aux dépendances de données, quel est l'ordre des opérations qui minimise le temps d'exécution ? Pour cela, nous avons construit une modélisation exacte du problème par programmation linéaire en variables 0-1. Notre premier modèle supposait un cache non bloquant, avec capacité non bornée de requêtes en attente, et un ensemble de tâches d'accès reliées en DAG (Directed Acyclic Graph). Nous avons ensuite étendu ce modèle au cas d'un cache non bloquant avec capacité bornée. Dans le cas de boucles, le comportement des accès n'est pas 1-périodique, il faut donc dans une phase préliminaire déterminer la périodicité. Durant son stage de DEA, Tarek Sahli a construit ces modèles et, partant d'un graphe (acyclique) de tâches, a implémenté la génération automatique du programme linéaire correspondant. Ce programme est ensuite résolu par le logiciel « CPLEX » [34].

6.3.3 Inclusion d'objets en Java

Participants : Patrick Carribault, Albert Cohen.

L'analyse de références en Java présentée précédemment permet d'envisager des transformations automatiques de programmes plus ambitieuses. Nous avons en effet exploré une extension

de la notion d'*inclusion d'objets* (*object inlining*, *unboxing*) pour Java, dans le cadre du stage de Patrick Carribault. Notre approche s'appuie sur la séparation entre objets dynamiques et variables, sur le *ramasse-miettes* automatique (*garbage collector*), et sur la présence de classes de *conteneurs* de haut niveau. Nous avons montré l'intérêt de l'inclusion d'objets sur de petits exemples, en reprenant et en étendant le modèle introduit par Julian Dolby [DC00]. Nous avons expérimenté un modèle d'annotations du bytecode Java (avec Kaffe), avec quelques résultats expérimentaux encourageants. La formalisation de critères sémantiques pour l'inclusion est en cours, en s'intéressant notamment aux transformations de conteneurs (aplatissement de tableaux, transformations de listes en tableaux, élimination de pointeurs dans les arbres, etc.). Deux nouveaux stages débutent sur le sujet, avec le développement d'une machine virtuelle traitant des directives d'inclusion automatique (utilisant OpenJIT).

6.4 Optimisation des programmes - registres

6.4.1 Allocation des registres dans les boucles

Participants : Christine Eisenbeis, Sid Ahmed Ali Touati.

Nos études passées sur l'allocation des registres dans les boucles, basées sur le meeting graph, ont donné lieu à des recherches plus théoriques, où l'on étudie les relations entre le degré chromatique du graphe d'interférences et le nombre de circuits dans le meeting graph [7].

Nous travaillons désormais sur une extension du meeting graph. Au lieu de partir d'un ordonnancement fixé sur le graphe d'interférences, nous étudions directement l'influence de l'allocation des variables dans les registres, sur la performance du code attendue, en l'occurrence sur le chemin critique du graphe de dépendances.

La relation de « meet » dans le meeting graph est remplacée par la relation « reuse ». Ces contraintes de réutilisation (d'un même espace mémoire pour des variables temporaires différentes) freinent le débit attendu de la boucle (caractérisé par l'intervalle d'initialisation II). On peut alors, en jouant sur les distances de réutilisation μ placées sur les flèches de « reuse », contrôler la pression en registres ρ qui se trouve être la somme des μ, en fonction de II [13].

Grâce à cette étude, nous pouvons donner des bornes sur l'intervalle d'initialisation pour le pipeline logiciel définies par les contraintes de registres. De plus, nous pouvons assurer avant la phase d'ordonnancement que nous avons toujours assez de registres pour la phase d'allocation sans nécessiter de code de vidage (*spill code*).

Nous avons ensuite prouvé que si le schéma de réutilisation est fixé, alors la minimisation du nombre de registres nécessaires pour un intervalle de lancement II donné est un problème polynomial.

Nous avons aussi donné une formulation exacte par programme linéaire en nombres entiers du problème général (schéma de réutilisation non fixé). Cette formulation est générée automatiquement à partir d'un graphe de dépendances reliant les opérations d'un corps de boucle. Les expériences montrent que, malgré la complexité théorique, cette approche est viable pour

[DC00] J. Dolby, A. A. Chien, « An evaluation of automatic object inline allocation techniques », *in: ACM Symp. on Programming Language Design and Implementation (PLDI'00)*, Vancouver, British Columbia, Canada, juin 2000.

des petites tailles. Pour les plus grandes tailles, on peut jouer avec les heuristiques de CPLEX, et aussi borner la durée du calcul ou la taille mémoire nécessaire au calcul [35].

6.5 Optimisation des programmes - modèles et techniques d'ordonnancement

6.5.1 Ordonnancement des Réseaux de Processus de Kahn

Participants : Paul Feautrier, Albert Cohen, Christine Eisenbeis, François Thomasset.

Un réseau de processus de Kahn (KPN) est composé d'un ensemble fini de processus qui communiquent entre eux par des canaux FIFO disposant d'une capacité de stockage infinie. Chaque processus exécute un programme séquentiel pouvant utiliser des instructions de lecture (bloquante) ou d'écriture (non bloquante) sur un canal spécifié. De plus, un KPN doit respecter la règle suivante : un canal ne peut être connecté qu'à un seul processus en entrée et à un seul processus en sortie. Si le comportement des processus est déterministe, alors G. Kahn a montré que l'histoire de chaque canal est également déterministe. L'intérêt des KPN en synthèse de systèmes embarqués est qu'ils permettent de décrire des systèmes réactifs comportant du parallélisme, tout en se prêtant à une analyse « par dépendances » analogue à celle qui permet de traiter les codes séquentiels. Ils conduisent d'autre part à une représentation graphique qui est familière aux spécialistes du traitement du signal et de l'électronique.

La principale question que l'on doit se poser au sujet d'un KPN est celle de la bornitude des canaux, qui conditionne évidemment la faisabilité du système. Un autre aspect est celui du degré de parallélisme, qui pour un KPN, est de l'ordre du nombre de processus. Il est bien rare que ce degré corresponde du premier coup à celui du matériel utilisé pour l'implémentation. Il peut être aussi bien trop élevé que trop faible ; il n'est donc pas paradoxal de parler de *parallélisation* d'un KPN.

Dans le cadre de notre contrat de recherche avec la société Philips, nous avons développé une méthode d'ordonnancement des KPN. La partie qui concerne les dépendances de données à l'intérieur d'un processus est standard. Par contre, il faut introduire des *dépendances de messages* : un message ne peut être reçu avant d'être émis. On y parvient en numérotant les émissions et les réceptions sur un même canal. La réception numéro n doit alors être ordonnancée *après* l'émission n. On peut écrire de la même façon la contrainte qui exprime que la taille du canal est bornée. Soulignons que le numérotage est virtuel et n'entraîne pas d'estampillage des messages. Sous des hypothèses usuelles (caractère affine des bornes de boucles) bien vérifiées en traitement du signal, le numérotage se ramène au calcul du nombre de points entiers dans un polyèdre paramétrique, problème bien résolu actuellement (Ph. Clauss et al.). De plus, dans les cas que nous rencontrons en pratique, des méthodes de comptage plus simples semblent largement suffisantes. Ce comptage effectué, les méthodes usuelles d'ordonnancement (par exemple l'utilisation du lemme de Farkas) s'appliquent directement.

Cependant, la méthode ci-dessus ne prend pas en compte les limitations des ressources disponibles (processeurs, opérateurs, mémoire). On peut remédier a ce défaut par des méthodes *ad hoc*, par exemple en simulant les contraintes de ressources par des dépendances de données. Ceci conduit à traiter des indices contenant des opérations modulo. Nous y sommes parvenus en construisant (partiellement) la coque entière de la relation de dépendance.

Un prototype d'ordonnanceur (Yaka) a été réalisé en Maple et va être incessamment porté en MuPad. Il utilise le logiciel PIP (Parametric Integer Programming) pour le calcul des dépendances et l'ordonnancement. A plus long terme, nous étudions plusieurs méthodes permettant de prendre en compte directement les contraintes de ressources.

Une fois l'ordonnancement obtenu, il faut en déduire le programme parallèle associé. Si le problème est maintenant bien compris pour une architecture « abstraite », nous devrons, pour des raisons d'efficacité, adapter la solution générale au cas particulier des architectures utilisées en traitement du signal.

6.5.2 Pipeline logiciel sur processeurs à bancs de registres multiples (« Clustered processors »)

Participants : Albert Cohen, Noureddine Djaballah, Christine Eisenbeis.

La compartimentation des registres en bancs est nécessaire pour augmenter le nombre de données accessibles en parallèle dans un processeur. Cette technique est donc adoptée dans de nombreuses versions récentes des DSP (processeurs de traitement du signal). Nous avons étudié le problème de savoir si une modélisation adéquate de cette spécificité pouvait se ramener à un cas classique, auquel cas nos outils de pipeline logiciel étaient directement utilisables.

Ce type d'architecture se présente sous la forme de plusieurs « processeurs » (ensemble d'unités fonctionnelles) qui communiquent par l'intermédiaire d'instructions de recopie entre bancs de registres. Le problème de pipeline se décompose donc en 2 problèmes, un problème de placement des opérations sur les « processeurs », puis l'ordonnancement proprement dit. Dans son stage de DEA, Noureddine Djaballah a choisi d'utiliser la méthode de partitionnement de Kim et Brown pour une parallélisation *a priori*. Les différentes itérations d'une même tâche sont exécutés sur le même processeur. Le nouveau graphe, avec les instructions de copie entre bancs de registres, est ensuite généré automatiquement pour servir d'entrée au logiciel PiLo [31].

7 Contrats industriels (nationaux, européens et internationaux)

Nous listons ci-dessous les partenaires impliqués dans nos actions industrielles.

- l'ONERA, France est partenaire dans le projet ESPRIT MHAOTEU ; ils nous fournissent des programmes à optimiser sur les architectures COMPAQ/Digital-alpha (section 8.2.1) ;
- Analog Devices, Edinburgh, Royaume-Uni est partenaire dans MHAOTEU (section 8.2.1) ;

Philips, Pays-Bas était partenaire du projet ESPRIT OCEANS, de compilateur pour le processeur TriMedia. Nous continuons à collaborer avec leur équipe de compilation dans le cadre d'un projet avec le LEP (Laboratoire d'Electronique de Paris) de Philips, devenu PRF (Philips Research France) et le labo Philips NatLab d'Eindhoven. Daniela Genius, de Philips, est collaborateur extérieur dans A3.

Le sujet de la collaboration est la définition et l'analyse de langage permettant le pilotage de processeurs graphiques et vidéo. L'avènement de la télévision numérique entraîne en effet la convergence des domaines de traitements graphiques et vidéo vers un cadre commun, où les flots d'images provenant de sources diverses (canaux de télévision, pages web, etc) seront intégrés

en vue de leur restitution sur un récepteur à haute résolution. Les traitements consistent par exemple à changer l'échelle d'un flot d'images, ou à le composer avec des images graphiques ; ils sont soumis à de fortes contraintes de bande passante et de temps réel. Le problème posé concerne la construction d'environnements permettant la programmation de telles unités de traitement, et si possible l'élaboration de compilateurs reciblables. Notre collaboration porte sur les points suivants :

- définition précise des besoins ;
- définition d'un langage et des étapes de compilation ;
- modélisation des débits de données, et prédiction des tailles des tampons nécessaires ;
- adaptation de nos outils d'ordonnancement à la génération de code pour ces processeurs.

8 Actions régionales, nationales et internationales

8.1 Actions nationales

Véronique Donzeau-Gouge, professeur au CNAM, est directeur officiel de la thèse de Pierre Amiranoff, qui participe aux réunions de son groupe BIP, autour des spécifications formelles. Catherine Dubois participe aussi à l'encadrement de Pierre Amiranoff.

Pour le calcul des fenêtres généralisées (voir 6.2.2), nous collaborons avec Philippe Clauss de l'Université de Strasbourg, qui est directeur officiel de la thèse d'Andry Randrianatoavina.

A3 organise un séminaire commun avec le groupe CRI (Centre de Recherches en Informatique) de l'École des Mines de Paris et le LRI de l'Université d'Orsay :

- 9 Janvier 2001 : *Élimination de redondances pour achitectures EPIC*, par Ivan Djelic (Laboratoire PRiSM, Université de Versailles)
- 6 Février 2001 : *Une suite de variations chromatiques*, par Dominique de Werra (Ecole Polytechnique Fédérale de Lausanne)
- 15 Février 2001 : *High Performance Computing and Trends : Connecting Computational Requirements with Computing Resources*, par Jack Dongarra (University of Tennessee)
- 19 Mars 2001 : *Analyse de programmes probabilistes par interprétation abstraite*, par David Monniaux (Laboratoire d'informatique de l'École Normale Supérieure, Paris)
- 21 Mars 2001 : *Gestion dynamique de flots dans le projet MPA*, par Smaïl Niar, (Université de Valenciennes)
- 25 Avril 2001 : *Prefetching from main memory*, par Jean-Loup Baer (Université de Washington)
- 5 Octobre 2001 : *Prototyping High-Performance Programs in a Functional Programming Language*, par Christian Lengauer (Universität Passau)
- 16 Octobre 2001 : *Le modèle de parallélisme par 'blob'*, par Frédéric Gruau (LRI, Orsay)
- 16 Octobre 2001 : *Reconnaissance d'algorithmes*, par Denis Barthou (PRiSM), Paul Feautrier (INRIA et PRiSM), et Xavier Redon (Université des Sciences et Technologies de Lille)

8.2 Actions européennes

8.2.1 MHAOTEU

Participants : Aloke Bajpai, Cédric Bastoul, Christine Eisenbeis, Paul Feautrier, Andry Randrianatoavina, François Thomasset, Sid Ahmed Ali Touati.

Le projet ESPRIT LTR MHAOTEU (Memory Hierarchy Analysis and Optimization Tools for the End-User) rassemble, outre l'INRIA, l'Université d'Edinburgh (Royaume-Uni), l'Université Polytechnique de Catalogne (Espagne) et l'Université de Versailles–Saint-Quentin, ainsi que Analog Devices, ex-EPC (Edinburgh Portable Compilers, UK) et l'ONERA (France). Le projet MHAOTEU vise l'étude et le développement d'outils d'analyse et d'optimisation de la hiérarchie mémoire pour les utilisateurs. Il a débuté en décembre 1997 et s'est conclu en mars 2001 [25, 29, 26, 33].

8.2.2 Autres collaborations

Dans le cadre d'un contrat PROCOPE (France-Allemagne), nous collaborons avec l'équipe de Christian Lengauer, de l'Université de Passau. Deux nouveaux sujets de recherches sont en cours : l'amélioration des techniques polyédriques d'ordonnancement (pavage, décomposition, ressources), et la reconnaissance d'algorithmes sur des structures de données avec pointeurs. Nous avons reçu le professeur Chris Lengauer et Niels Elmenreich du 3 au 8 octobre 2001, ainsi que Martin Griebl du 24 au 28 septembre 2001, et du 29 mars au 3 avril 2001. Paul Feautrier était à Passau du 18 au 22 juin 2001.

Denis Barthou a passé 4 semaines à Passau en automne 2001, et Albert Cohen y a passé 5 jours pendant cette période.

8.3 Actions internationales

Dans le cadre d'un contrat entre le CNRS et l'UIUC (Université de l'Illinois à Urbana-Champaign), Albert Cohen a passé trois mois supplémentaires dans le laboratoire de David Padua (hiver 2001) et Paul Feautrier était à l'UIUC du 7 au 12 mai.

Au titre d'un contrat CMCU, Commission Mixte (franco-tunisienne) pour la Coopération Universitaire), nous collaborons avec l'équipe du professeur Zaher Mahjoub (Mohamed Jemni et Yosr Slama) de la Faculté des Sciences de Tunis. Yosr Slama nous a rendu visite du 15 juin au 15 août 2001, pour travailler sur la génération de code parallèle.

Dans le cadre d'une collaboration NSF-INRIA entre l'Université de South California (USC) de Los Angeles, l'Université de Delaware et le projet A3, Sid Ahmed Ali Touati a passé le mois de février 2001 au département Electrical Engineering dans l'equipe du Professeur Jean-Luc Gaudiot afin de réfléchir à certains problèmes au sujet de la compilation d'applications sur architectures multi-threads.

8.4 Visites et invitations de chercheurs

Les accueils de chercheurs extérieurs sont indiqués dans les sections 8.1, 8.2 et 8.3.

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

Ch. Eisenbeis était rapporteur de la thèse de Guillaume Huard (Ecole Normale Supérieure de Lyon, 6 décembre 2001 et faisait partie du jury de J. Janssen (Université de Delft, Pays-Bas, 17 septembre 2001). Ch. Eisenbeis a fait partie des comités de programme de CC' 01, PLDI' 2001 et EuroPar' 2001.

Paul Feautrier fait partie des comités de rédaction des journaux : « IEEE Transactions on Parallel and Distributed Systems », « International Journal of Parallel Programming », et « Parallel Computing ». Il était dans le jury d'habilitation de Nahid Emad (15 février 2001, Université de Versailles-Saint-Quentin), et les jurys de thèse de Stefan Balev (21 mars 2001, Amiens), Peng Wu (11 mai 2001, Urbana-Champaign (USA)) et Nicolas Museux (20 décembre 2001, Ecole des Mines de Paris).

9.2 Enseignement universitaire

Albert Cohen est chargé de cours à l'Université de Versailles (C et systèmes d'exploitation), en DEA à l'Université Paris-Sud (Compilation pour architectures à hautes performances), et chargé de TD avec Olivier Temam à l'école Polytechnique (Architecture).

Pierre Amiranoff est 1/2 ATER. Il anime des travaux dirigés en Algorithmique-Programmation cycle A du CNAM. Le langage-support est ADA_95.

Albert Cohen est chargé de cours et TD à l'Université de Versailles (C et systèmes d'exploitation).

Christine Eisenbeis a assuré un cours de 3ème année à l'ENSTA, sur la conception de logiciels pour processeurs embarqués.

Christine Eisenbeis et François Thomasset sont intervenus dans le cours de Compilation Avancée du DEA d'Orléans.

Paul Feautrier enseigne au MISI (UVSQ) et SIR (UPMC).

9.3 Participation à des colloques, séminaires, invitations

Les membres du projet ont donné les séminaires suivants :

- Workshop sur l'« Interaction between Compilers and Computer Architectures », INTERACT-5, Monterey, Mexique, 20-24 janvier 2001 (Sid Ahmed Ali Touati, « EquiMax : A New Formulation of Acyclic Scheduling Problem for ILP Processors »).
- Séminaire au DCS de l'Université de l'Illinois à Urbana Champaign (Albert Cohen, « Pointer Analysis for Monotonic Container Traversals »), mars 2001.
- Journées de l'ASTI, Paris, avril 2001 : séminaire « prix SPECIF 2000 » (Albert Cohen, accessit), et participation aux conférences RenPar et Sympa (Cédric Bastoul, Albert Cohen, Pierre Amiranoff, Christine Eisenbeis, Paul Feautrier).
- Conférence ETAPS, « CC2001, Compiler Construction » (Christine Eisenbeis, présentation du papier de Sid Ahmed Ali Touati : « Register Saturation in Superscalar and VLIW Code Case of Direct Acyclic Data Dependence Graphs »), Gênes, Italie, 3 avril 2001.

- Présentation à « ACM International Conference on Supercomputing » (ICS'01), Sorrente, Italie (Albert Cohen, « Monotonic Evolution : an Alternative to Induction Variable Substitution for Dependence Testing »), juin 2001.
- Présentations au workshop « Compilers for Parallel Computers » (CPC'01), Edimbourg, Royaume-Uni (Albert Cohen, « Dependence Testing without Induction Variables substitution », Daniela Genius, « A Case for Array Merging in Memory Hierarchies » et Christine Eisenbeis, « SIRA : Schedule Independent Register Allocation for Software Pipelining »), juin 2001.
- Participation à la conférence « Static Analysis Symposium » (SAS'01), Paris (Albert Cohen), juillet 2001.
- Séminaire au PRiSM, Université de Versailles St-Quentin (Patrick Carribault, soutenance de stage « Autour de l'inclusion d'objets en Java »), septembre 2001.
- Séminaire Philips Research France, Suresnes (Daniela Genius, Albert Cohen et soutenance de stage d'Abdesselem Kortebi « Réseaux de processus hiérarchiques : motivations, modèle et implémentation »), octobre 2001.
- Séminaire A3/CRI/LRI (Paul Feautrier, « Reconnaissance d'algorithmes »), 16 octobre 2001.
- Séminaire au départment d'informatique et mathématiques (FMI) de l'Université de Passau, (Albert Cohen, « Pointer Analysis for Container-Centric Applications »), novembre 2001.
- Exposé à TU Delft (Daniela Genius, « Hierarchical Process Networks »), novembre 2001.
- Faculté des Sciences de Tunis (Paul Feautrier, « Algorithm recognition »), 2 novembre 2001.
- Séminaire Philips Research National Laboratories, Eindhoven, Pays-Bas (Albert Cohen, invité par l'équipe de Philips Research France, « Hierarchical Process Networks : modeling and verifying stream-processing applications »), décembre 2001.
- Colloquium du LCS (MIT) (Paul Feautrier, « Algorithm recognition »), 7 décembre 2001.

10 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] A. Cohen, *Program Analysis and Transformation : from the Polytope Model to Formal Languages / Analyse et transformation de programmes : du modèle polyédrique aux langages formels*, thèse de doctorat, Université Versailles–Saint-Quentin-en-Yvelines, dec 1999.

[2] J.-F. Collard, D. Barthou, P. Feautrier, « Fuzzy array dataflow analysis », *in : Proc. of 5th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, Santa Barbara, CA, jul 1995.

[3] C. Eisenbeis, W. Jalby, A. Lichnewsky, « Compiler techniques for optimizing memory and register usage on the CRAY2 », *International Journal on High Speed Computing 2*, 2, 1990, p. 193–222, appeared also as INRIA Research Report no 1302 October 1990.

[4] C. Eisenbeis, W. Jalby, D. Windheiser, F. Bodin, « A Strategy for Array Management in Local Memory », *Mathematical Programming 63*, 1994, p. 331–370, Special Issue on Applications of Discrete Optimization in Computer Science.

[5] C. Eisenbeis, S. Lelait, « LoRA : a Package for Loop Optimal Register Allocation », *Rapport de recherche n° 3709*, INRIA, Rocquencourt, juin 1999, `http://www.inria.fr/rrrt/rr-3709.html`.

[6] P. Feautrier, « Dataflow Analysis of Scalar and Array References », *Int. J. of Parallel Programming 20*, 1, février 1991, p. 23–53.

[7] S. Lelait, *Contribution à l'allocation de registres dans les boucles*, Thèse de doctorat, Université d'Orléans, jan 1996.

[8] K. S. McKinley, O. Temam, « A Quantitative Analysis of Loop Nest Locality », *in : ASPLOS' 96*, Cambridge, Massachussets, oct 1996.

[9] A. Sawaya, *Pipeline Logiciel : Découplage et Contraintes de Registres*, thèse de doctorat, Université de Versailles - INRIA Rocquencourt, 1997.

[10] J. Wang, C. Eisenbeis, M. Jourdan, B. Su, « DEcomposed Software Pipelining : a New Perspective and a New Approach », *International Journal on Parallel Processing 22*, 3, 1994, p. 357–379, Special Issue on Compilers and Architectures for Instruction Level Parallel Processing.

Communications à des congrès, colloques, etc.

[11] A. Cohen, P. Wu, « Dependence Testing Without Induction Variable Substitution », *in : Workshop on Compilers for Parallel Computers (CPC'01)*, Edinburgh, Scotland, UK, juin 2001, `http://www.icsa.informatics.ed.ac.uk/cpc2001/Proceedings/cohen.ps`.

[12] D. Genius, S. Lelait, « A Case for Array Merging in Memory Hierarchies », *in : Workshop on Compilers for Parallel Computers (CPC'01)*, Edinburgh, Scotland, UK, juin 2001.

[13] S.-A.-A. Touati, C. Eisenbeis, « SIRA : Schedule Independent Register Allocation for Software Pipelining », *in : Workshop on Compilers for Parallel Computers (CPC'01)*, Edinburgh, Scotland, UK, juin 2001, `http://www.icsa.informatics.ed.ac.uk/cpc2001/Proceedings/eisenbeis.ps`.

[14] S.-A.-A. Touati, « EquiMax : A New Formulation of Acyclic Scheduling Problem for ILP Processors », *in : INTERACT-5 : Workshop on Interaction between Compilers and Computer Architectures*, Kluwer Academic Publishers, Monterrey, Mexique, janvier 2001.

[15] S.-A.-A. Touati, « Maximizing for Reducing Register Need in Acyclic Schedules », *in : Proceedings of 5th International Workshop on Software and Compilers for Embedded Systems, SCOPES*, St Goar, Germany, mars 2001.

[16] S.-A.-A. Touati, « Optimal Acyclic Fine-Grain Schedule with Cache Effects for Embedded and Real Time Systems », *in : Proceedings of 9th nternational Symposium on Hardware/Software Codesign, CODES*, ACM, Copenhagen, Denmark, avril 2001.

[17] S.-A.-A. Touati, « Register Saturation in Superscalar and VLIW Codes », *in : Proceedings of The International Conference on Compiler Construction, Lecture Notes in Computer Science*, Springer-Verlag, Genova, Italy, avril 2001.

[18] P. Wu, A. Cohen, D. Padua, J. Hoeflinger, « Monotonic Evolution : an Alternative to Induction Variable Substitution for Dependence Analysis », *in : ACM Int. Conf. on Supercomputing (ICS'01)*, Sorrento, Italy, juin 2001.

[19] P. Wu, A. Cohen, D. Padua, « Induction Variable Analysis Without Idiom Recognition : Beyond Monotonicity », *in : Workshop on Languages and Compilers for Parallel Computing (LCPC'01)*, Cumberland Falls, Kentucky, USA, août 2001.

Rapports de recherche et publications internes

[20] P. Amiranoff, A. Cohen, P. Feautrier, « Variables d'induction généralisées pour l'analyse par instances de programmes récursifs », *Rapport de recherche n°4252*, INRIA, septembre 2001, `http://www.inria.fr/rrrt/rr-4252.html`.

[21] D. Barthou, P. Feautrier, X. Redon, « On the Equivalence of Two Systems of Affine Recurrence Equations », *rapport de recherche n°4285*, INRIA, octobre 2001, `http://www.inria.fr/rrrt/rr-4285.html`.

[22] A. Cohen, P. Wu, D. Padua, « Intraprocedural Pointer Analysis for Container-Centric Applications », *rapport de recherche n°4289*, INRIA Rocquencourt, France, octobre 2001, `http://www.inria.fr/rrrt/rr-4289.html`.

[23] A. Cohen, P. Wu, D. Padua, « Pointer Analysis for Monotonic Container Traversals », *rapport de recherche n°1586*, CSRD, University of Illinois at Urbana-Champaign, USA, 2001.

[24] S.-A.-A. Touati, « Optimal Register Saturation in Acyclic Superscalar and VLIW Codes », *Research Report n°4263*, INRIA, septembre 2001, `http://www.inria.fr/rrrt/rr-4263.html`.

Divers

[25] J. Abella, C. Bastoul, J.-L. Béchennec, N. Drach, C. Eisenbeis, P. Feautrier, B. Franke, G. Fursin, A. Gonzales, T. Kishu, P. Knijnenburg, J. Llosa, M. O'Boyle, J. Sébot, X. Vera, « Guided Transformations », Deliverable M3.D2 of the MHAOTEU ESPRIT project n0 24942, janvier 2001.

[26] J. Abella, G. Fursin, A. Gonzalez, J. Llosa, M. O'Boyle, A. Prabhat, O. Temam, S.-A.-A. Touati, X. Vera, G. Watts, « Advanced Performance Analysis », Deliverable M3.D1 of the MHAOTEU ESPRIT project n0 24942, février 2001.

[27] E. Ayguadé, F. Dahlgren, C. Eisenbeis, R. Espasa, G. R. Gao, H. Muller, R. Sakellariou, A. Seznec, « Instruction-Level Parallelism and Computer Architecture », LNCS 2150, *in : EuroPar' 2001*, 2002, Introduction to EuroPar Topic 08+13.

[28] N. Bansal, « Software Pipelining and PiLo ALISE Interface », Rapport de stage, INRIA Rocquencourt, juillet 2001.

[29] P. d'Anfray, C. Eisenbeis, « MHAOTEU Tools : Installation Guide and Getting Started Example », Deliverable M3.D3 of the MHAOTEU ESPRIT project n0 24942, janvier 2001.

[30] D. de Werra, C. Eisenbeis, S. Lelait, E. Stöhr, « Circular arc graph coloring : on chords and circuits in the meeting graph », European Journal of Operational Research, *136*, 2002, à paraître.

[31] N. Djaballah, « Optimisation de code pour processeurs VLIW à bancs de registres multiples », Rapport de DEA, Université d'Orléans, septembre 2001.

[32] I. Djelic, « Élimination de redondances partielles pour architectures EPIC », 2002, à paraître.

[33] P. Guillen, E. Garnier, S.-A.-A. Touati, « End-User Application : ONERA's Aerodynamic Solver FLU3M », Deliverable M3.D3 of the MHAOTEU ESPRIT project n0 24942, février 2001.

[34] T. Sahli, « Modèle linéaire pour minimiser les défauts du cache », Rapport de DEA, Université d'Orléans, septembre 2001.

[35] S.-A.-A. Touati, C. Eisenbeis, « Cyclic Register Pressure and Allocation for Modulo Scheduled Loops », Rapport de recherche INRIA, 2002, à paraître.