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 CNRS

Jean-François Collard [CR CNRS, jusqu'au 31 janvier 2000]

Personnel PRISM, Université de Versailles–Saint-Quentin

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

Paul Feautrier [Professeur, jusqu'au 30 septembre 2000]

Collaborateurs extérieurs

Daniela Genius [Bourse Marie Curie, LEP Philips, à partir du 18 octobre 2000]

Olivier Temam [Professeur à l'Université d'Orsay]

Chercheurs post-doctorants

Daniela Genius [Bourse INRIA, du 1er septembre au 17 octobre 2000]

Doctorants

Pierre Amiranoff [professeur certifié de mathématiques, détaché comme 1/2 ATER IIE(Evry)-CNAM]

Cédric Bastoul [bourse MENRT, Université de Paris VI, à partir du 1er septembre 2000]

Albert Cohen [AMN, Université de Versailles-Saint-Quentin, jusqu'au 31 août 2000]

Min Dai [Université de Versailles–Saint-Quentin, jusqu'au 9 mai 2000]

Ivan Djelic [bourse MENRT, Université de Paris VI]

Alexandre Farcy [bourse MENRT, Université de Paris VI]

Ping Hu [bourse INRIA, Université de Paris VI, jusqu'au 31 juillet 2000]

Andry Randrianatoavina [bourse MENRT, Université de Strasbourg]

Sid Ahmed Ali Touati [bourse CROUS France/Algérie]

Grégory Watts [contrat ESPRIT MHAOTEU, Université de Versailles–Saint-Quentin]

Stagiaires

Aloke Bajpai [Indian Institute of Technology, Delhi, Inde, du 1er mai au 30 juin 2000]

Cédric Bastoul [stage de DEA, Université de Paris VI, du 1er mars au 31 août 2000]

Abishek Prabhat [Indian Institute of Technology, Delhi, Inde, du 1er mai au 30 juin 2000]

Arjan Salomons [Université de Leiden, Pays-Bas, du 1er mai au 30 juin 2000]

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

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 réduit alors à un problème de modélisation des contraintes.

[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.

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 ;
- architecture de processeur.

Le projet ESPRIT OCEANS, achevé fin 1999, s'articulait justement autour de l'interaction entre les 2 phases de pre-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 [11]) ainsi que dans l'environnement SALTO (projet ESPRIT OCEANS).

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, aprés 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 [11]. Il utilise l'environnement Sage++ de manipulation de programmes.

6 Résultats nouveaux

6.1 Analyse sémantique des programmes

6.1.1 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 très 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 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.

En collaboration avec Peng Wu et David Padua (de l'Université de l'Illinois à Urbana-Champaign), le problème a été attaqué de trois façons.

- Une première méthode consiste à associer à chaque référence l'ensemble des objets qu'elle peut désigner. La difficulté est le traitement des tableaux de références, qui sont omniprésents en Java. Il est indispensable d'individualiser chaque cellule du tableau si on veut pouvoir faire, par exemple, une analyse de dépendances. Le calcul des ensembles pointés se mène par une classique recherche itérative de points fixes. Une implémentation pilote a été réalisée à l'UIUC, avec des résultats très satisfaisants.

- On peut également tenter de trouver, dans les cas favorables, une information plus précise : l'unique objet pointé, à un instant donné, par chaque référence du programme. On peut pour cela écrire un système d'équations aux références, qu'il s'agit ensuite de résoudre. On y parvient en combinant des techniques gaussiennes et des solveurs élémentaires (calcul du flot des données, résolution de récurrences). Ici encore une implémentation pilote a été réalisée. Le travail en cours est l'exploration des limites de cette implémentation.

- Enfin, dans le cas de structures de données plus générales, les *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 procédér 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.

Contrairement à l'usage en matière d'analyses de pointeurs ou de références, les trois techniques partagent un formalisme de description des programmes capable de différencier les propriétés des objets à chaque instance (ou itération) d'une instruction à l'exécution. Il en résulte une précision supérieure et la possibilité d'effectuer des transformations de code plus radicales. La première méthode peut traiter des programmes sans aucune contrainte, mais les résultats obtenus sont approximatifs. La deuxième méthode, au contraire, donne des résultats précis pour certains programmes et aucun résultat pour d'autres. Il est clair qu'une synthèse est nécessaire. La troisième est à la fois plus générale, plus rapide et moins précise, et les travaux futurs porteront sur les aspects interprocéduraux.

6.1.2 Traduction de code gardé

Participant : Ping Hu.

Dans ce travail [27, 12], nous proposons un algorithme de traduction qui peut convertir un graphe de flot de contrôle en code gardé. La différence principale entre cette traduction et la plupart des autres algorithmes de conversion de branchements se trouve dans la définition de

[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.

la valeur de la garde associée à chaque bloc de base. Plutôt que de trouver, pour chaque opération, l'expression logique concernant toutes les conditions possibles dans lesquelles l'opération devrait être exécutée, nous traduisons chaque opération concernant un branchement dans un bloc de base en une assignation, qui assigne la condition du branchement directement à la garde du bloc cible. En même temps, l'assignation sera exécutée sous le contrôle de la garde du bloc courant. De cette façon, l'exécution du bloc courant de base déterminera dynamiquement le prochain bloc de base à exécuter. En conséquence, des gardes sont définies dynamiquement, plutôt que par des expressions logiques statiques.

En outre, nous appliquons l'algorithme de traduction aux constructions de boucles. La difficulté rencontrée est de pouvoir détecter les boucles arbitraires dans un graphe donné de flot de contrôle. Les approches existantes doivent habituellement découvrir d'abord les arcs de retour ("*back-edge*s"), puis identifier les boucles qui sont constituées par ces *back-edge*s. Cependant, la définition de *back-edge* est par convention basée sur des graphes de flots réductibles, plutôt que des graphes irréductibles de flots qui représentent les boucles avec entrées multiples. Malheureusement, les boucles dans des programmes en langage d'assembleur ont souvent plusieurs entrées. Afin de tenir également compte d'une telle sorte de boucles, nous définissons une nouvelle classe d'arcs, *loop-edge*s, qui causera des boucles impliquées dans un graphe de flot de contrôle, y compris les boucles avec entrées multiples.

De plus, nous prenons la peine de démontrer la correction de cet algorithme de traduction. Un algorithme de traduction est correct si le code traduit par l'algorithme préserve la signification du code initial. Des significations de programme doivent être formellement spécifiées par sémantique de programme. Bien que les spécifications de sémantiques pour des langages d'assembleur (voire langage machine) soit une question ouverte, nous essayons quand même d'aller dans cette direction. Nous utilisons la sémantique opérationnelle structurale pour représenter les significations des transferts de branchements dans les codes séquentiels et dans les codes gardés ; ainsi nous avons pu prouver la correction de l'algorithme traduisant les opérations de branchements dans un code séquentiel en opérations gardées.

6.1.3 Analyse des codes assembleur

Participant : François Thomasset.

Dans les compilateurs, l'analyse de flot de données est traditionnellement réalisée sur le programme source et exploitée dans les différentes étapes menant à la génération du code. L'analyse de code assembleur, qui a des applications évidentes en *reverse engineering*, nous intéresse pour deux raisons. La première est l'analyse a priori des performances d'un code assembleur, la seconde est le réordonnancement des instructions et la réallocation de registres dans le code assembleur. Ces deux aspects requièrent une étude fine des conflits possibles entre différents accès mémoire. Pour analyser les références mémoire, nous effectuons une analyse statique calculant des expressions symboliques décrivant le contenu des registres utilisés pour indexer la mémoire. Pour éviter l'explosion des informations calculées, nous avons choisi d'insérer aux débuts des boucles des points de réinitialisation, où l'on suppose l'absence d'information. Cette technique [21, 13] permet d'analyser sans perte d'information les parties acycliques du graphe de flot de contrôle, et il nous faut examiner comment l'étendre ou en utiliser les

résultats pour analyser finement les dépendances dans les boucles. Cette étude est réalisée en collaboration avec l'Université Friedrich Schiller à Iéna (Allemagne), dans le cadre d'une action PROCOPE. Une implémentation a été réalisée dans l'environnement SALTO (System for Assembly Language Tools and Optimisation) développé dans le projet CAPS de l'IRISA, et nous connectons la sortie de cet outil à l'entrée de PILO (section 5.1).

6.1.4 Analyse des codes assembleurs: connection avec le programme source

Participants : Christine Eisenbeis, Abishek Prabhat, François Thomasset, Sid Ahmed Ali Touati.

Dans le cadre du projet ESPRIT MHAOTEU (section 8.2.1), nous cherchons à comprendre les relations entre les optimisations réalisées au niveau du code assembleur et celles réalisées par des transformations du code source. Pour cela, nous utilisons l'outil ATAC, interface graphique du transformateur de code SALTO (System for Assembly Language Tools and Optimization) développé dans le projet CAPS de l'IRISA. Dans ce genre d'outil, il est classiquement facile de retrouver la ligne de code source correspondant à une ligne de code assembleur (pourvu que le compilateur ait généré les informations adéquates). Le travail d'Abishek Prabhat a été d'étudier la liaison inverse : étant donné une ligne dans le code source, comment trouver la ou les instructions de code assembleur qui lui correspondent ? Abishek Prabhat a implémenté un tel module dans ATAC, en se basant sur les informations assembleur-code source, et a amorcé le travail de recherche plus général sur la détermination de la correspondance programme source-code assembleur, dans le cas de code optimisé, et quand le compilateur n'a pas généré les informations [36].

Toujours dans le cadre du projet ESPRIT MHAOTEU (section 8.2.1), où nous travaillons principalement sur le code source, nous faisons des expériences pour vérifier que les analyses réalisées sur le code source donnaient des résultats en accord avec les analyses réalisées sur le code assembleur.

6.1.5 Spécifier les programmes et leurs propriétés

Participant : Pierre Amiranoff.

Les modalités d'intégration des propriétés mathématiques relatives aux données ou aux algorithmes d'un programme au sein de celui-ci en vue de l'optimisation constituent une question largement ouverte. Les interrogations concernent les points suivants :

- Quelles méthodes formelles, quels langages utiliser ? A priori, une décision sur ces questions ne peut survenir que comme sous-produit des 2 questions intimement liées qui suivent.
- A quels niveaux du processus global de construction de programme s'attacher : programme source, code intermédiaire, génération de code ?
- Quelles techniques d'optimisation mettre en œuvre : organisation du code source, informations fournies au compilateur, conception de compilateur ou de préprocesseur spécifiques ?

Le dialogue que nous voulons construire entre le programme-source et le compilateur peut prendre des formes très différentes :
2 types de critères doivent être évalués :

- on peut imaginer que la structure du programme source soit conçue pour offrir au compilateur des procédés de calcul optimisés d'office ;
  A l'inverse, on peut rechercher une écriture de ce programme qui laisse le travail d'optimisation au compilateur.

- on étudie et on utilise les compilateurs optimisants existants ;
  A l'inverse, on peut s'attacher à dessiner un compilateur (un schéma de compilateur, une modification souhaitable) adapté spécifiquement à notre propos, par exemple pour implémenter une technique optimisante relative à un type d'information particulier mis en évidence par la spécification formelle.

Cette année, nous nous sommes intéressés à des programmes contenant des boucles parallélisables. Nous avons travaillé sur l'analyse de dépendances au sein de ces boucles. Dans de nombreux programmes, les index de tableau sont fonction des indices de boucle. Quand cette fonction est injective, les références de tableau concernées sont indépendantes au sein de la boucle. Il est souvent possible d'organiser les données d'un problème de façon à faire apparaître des relations injectives les liant. Ce principe est à la base des méthodes de coloriage pour des calculs parallèles.
On peut donc écrire ou transformer un programme source pour exploiter ces caractéristiques. Une partie de la tâche est alors de valider la correction de la transformation, des points de vue du respect de la sémantique et des possibilités de parallélisation [30].
Une autre voie de notre recherche en cours est le travail sur *l'analyse de dépendances au niveau des instances* [1] .

6.1.6 Élimination des redondances partielles (mise à jour)

Participants : Jean-François Collard, Ivan Djelic.

L'élimination des redondances partielles est une optimisation très courante dans les compilateurs de recherche et commerciaux, elle est donc à présent bien maîtrisée. Cependant, le mécanisme architectural dit "de prédication", c'est-à-dire l'exécution conditionnelle des instructions, rend caduques les techniques actuelles d'élimination. D'autre part, l'avènement de processeurs disposant de prédication chez les grands fondeurs, comme l'Itanium d'Intel, rend urgent la mise au point de techniques adaptées d'élimination de redondances partielles. Deux articles sur le sujet ont été publiés [28, 24], et un article soumis à un journal. Des expérimentations ont été menées dans l'environnement Trimaran (compilateur IMPACT et simulation d'architecture EPIC).

6.2 Analyse des comportements des programmes

6.2.1 Fenêtres de référence généralisées - approche statique

Participants : Christine Eisenbeis, Andry Randrianatoavina, François Thomasset.

Cette étude concerne l'optimisation de code pour la hiérachie mémoire, thème du projet ESPRIT MHAOTEU (section 8.2.1).

Pour estimer le bien-fondé d'une transformation de code, il faut avoir un critère facile à calculer statiquement ou à mesurer dynamiquement. Le nombre *de cache miss* est un tel critère, mais il est extrêmement complexe à calculer statiquement, et coûteux à mesurer. En particulier les *cache miss* dus aux conflits mémoire sur un même *cache set* sont les plus difficiles à contrôler. Martonosi et al. en ont donné une formulation comme nombre de points entiers dans une union de polyèdres. La complexité du calcul, déjà très grande dans le cas d'un cache à correspondance directe, explose dans le cas d'un cache associatif par ensembles. En tout état de cause, la formule n'est pas paramétrée par l'ordonnancement des tâches et ne permet donc pas a posteriori une optimisation directe de code.

Nous étudions une autre approche, moins complexe en théorie, qui n'est plus basée directement sur le calcul du nombre de *cache miss*, mais sur le nombre de variables en vie dans chaque *cache set*. Nous perdons de la précision puisque ces 2 nombres ne sont pas directement liés. En retour, nous gagnons une moindre complexité de calcul ainsi que la prise en compte dans la formule de l'ordonnancement.

Le calcul du nombre de *cache miss* dans chaque *cache set* est une extension de nos études passées des fenêtres de références [4], appliquées en chaque *cache set*. Nous avons donc commencé par réimplémenter le calcul des fenêtres de référence dans l'outil Polaris de parallélisation de code. Cette implémentation utilise le logiciel de calcul du nombre de points entiers dans les polyèdres développé par Philippe Clauss à l'Université de Strasbourg, qui n'existait pas lors de nos premières études. De nombreuses difficultés théoriques persistent, en particulier le calcul de la taille de la projection d'un polyèdre paramétré. Cette année, nous avons tenté une autre voie d'attaque, en travaillant sur la structure mémoire des lignes de cache. Au lieu de linéariser l'adresse de la ligne, ce qui donne lieu à des polyèdres dégénérés, nous conservons la structure multidimensionnelle des tableaux le plus tard possible, et avons un treillis de lignes, ou, plus loin, un treillis des lignes qui s'appliquent sur un set s fixé. Ceci nous a permis de traiter le cas de la réorganisation des tableaux en mémoire (*array padding*).

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

6.2.2 Fenêtres de référence généralisées - approche dynamique

Participants : Aloke Bajpai, Christine Eisenbeis, Andry Randrianatoavina, Sid Ahmed Ali Touati.

Dans le cadre du projet ESPRIT MHAOTEU 8.2.1, pour étudier la pertinence des GRW (fenêtres de référence généralisées), en dépit de la complexité de l'approche statique, nous avions développé l'outil DGRW d'évaluation dynamique des fenêtres. Le but était seulement de vérifier expérimentalement la corrélation entre la taille de la fenêtre et le nombre de *cache miss* dus aux interférences.

Cette année, nous avons développé un module pour les outils MHAOTEU, de visualisation de l'évolution au cours du temps de la fenêtre généralisée. Ce module est basé sur le même schéma que l'outil CVT (Cache Visualisation Tool) développé par Grégory Watts - serveur qui interprète une suite de références mémoire générées au vol. Cet outil, nommé DGVT (Dynamic General reference window Visualisation Tool) a été développé par Aloke Bajpai lors de son stage dans le projet [34]. De même que CVT, il peut être utilisé comme un *debugger* de performances.

6.2.3 Prédiction de branchement : les limites de la corrélation

Participants : Alexandre Farcy, Olivier Temam.

Les mécanismes de prédiction des branchements conditionnels les plus efficaces utilisent la corrélation entre branchements. Dans cette étude, nous définissons et classons les relations de corrélation complexes que ces mécanismes arrivent à extraire. Pour les branchements les plus mal prédits, nous montrons les limites de la prédiction par corrélation. Nous montrons également qu'il existe un potentiel pour calculer plus tôt leur condition. Enfin, nous proposons des méthodes alternatives, exploitant ce potentiel, permettant de réduire le risque de mauvaise prédiction [26].

6.3 Optimisation des performances - registres

6.3.1 Saturation en Registres dans les graphes de dépendances acycliques

Participant : Sid-Ahmed-Ali Touati.

Nous avons étudié dans [32] la notion de saturation en registres dans les graphes de dépendance de données acyclique. Elle consiste à déterminer la borne maximale exacte du besoin en registres d'un GDD indépendamment des ordonnancements et des unités fonctionnelles du processeur cible. Nous avons prouvé que ce problème est NP-complet et proposé une heuristique gloutonne qui donne des résultats quasi optimaux [33]. Notre étude théorique nous a permis de démontrer aussi que les travaux publiés dans [Ber96,BGSL93] sont erronés en ce qui concerne la détermination de la limite maximale du besoin en registres.

Calculer la saturation en registres permet de savoir si les contraintes de registres peuvent être ignorées lors de la phase d'ordonnancement de code. C'est le cas lorsque la saturation en registres est inférieure au nombre de registres disponibles R dans le processeur cible. Dans le cas contraire, nous insérons des arcs dans le GDD afin de réduire la saturation en dessous du seuil R tout en minimisant le chemin critique. Nous avons prouvé que ce problème est NP-hard et avons proposé dans [32, 33] une heuristique très efficace pour les GDD acylciques. Nos résultats sont soumis à une conférence internationale dans [38].

[Ber96] Berson, David A., *Unification of Register Allocation and Instruction Scheduling in Compilers for Fine-Grain Parallel Architecture*, thèse de doctorat, Pittsburgh University, 1996.

[BGSL93] D. Berson, R. Gupta, Soffa, M. L., « URSA: A Unified ReSource Allocator for Registers and Functional Units in VLIW Architectures», *in : Conference on Architectures and Compilation Techniques for Fine and Medium Grain Parallelism*, p. 243–254, Orlando, Florida, janvier 1993, `http://www.cs.pitt.edu/~~soffa/research/Comp/pact93.ps`.

6.3.2 Nouvelle Formulation des Problème d'Ordonnancement sous Contraintes de Registres et de Ressources avec des Effets de Cache

Participant : Sid-Ahmed-Ali Touati.

Nos travaux précédents nous ont fait aboutir à trouver une meilleure formulation du problème d'ordonnancement sous contraintes de registres et de ressources. Etant donné un GDD acyclique ayant n opérations et m dépendances, la complexité de notre programme linéaire en nombres entiers est de $O(n^2)$ variables et $O(m+n^2)$ contraintes. Cette complexité est meilleure que celles des techniques qui existent déjà dans le domaine. Nous avons soumis notre nouvelle formulation pour une publication dans [37]. Les effets de cache peuvent aussi être inclus en associant des latences variables pour les accès à la mémoire. Selon l'ordonnancement des opérations mémoire, cette latence devient ou bien une pénalité d'accès à la mémoire si c'est un défaut de cache, ou bien un accès direct au cache. Pour l'instant, seuls les *compulsory miss* sont pris en compte. Notre formulation sera soumise prochainement pour publication.

6.3.3 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]. Ces recherches se poursuivent[35] dans notre groupe de travail (D. de Werra, Ch. Eisenbeis, S. Lelait, E. Stöhr), voir section 8.2.

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.

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*.

6.4 Optimisation des performances - localité des données

6.4.1 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 défaults 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 traffic et d'utiliser cette information pour calculer un découpage optimum. Le découpage peut donner lieu, le cas échéant, à des restructurations du programme source. Une implémentation pilote, qui couvre des cas simples, a été réalisée par Cédric Bastoul dans le cadre de son stage pour le DEA "Systèmes Informatiques Répartis" de l'université P. et M. Curie. Cédric a obtenu une bourse de thèse qui doit lui permettre de mener cette étude à son terme.

L'étape suivante est la transposition de ces recherches aux mémoires locales qui remplacent souvent les caches sur les processeurs embarqués. Un constructeur de systèmes embarqués a déjà manifesté de l'intérêt pour cette extension.

6.4.2 Ordonnancement d'instructions de chargements en utilisant le profiling

Participants : Goëtz Lindenmaier, Kathryn S. McKinley, Olivier Temam.

Dans cette étude, nous montrons comment utiliser l'information fournie par les compteurs matériels d'une architecture (en l'occurrence un processeur Alpha). Nous modifions l'algorithme d'ordonnancement de Lo et Egger et améliorons sa capacité à ordonnacer des instructions de chargement à latence variable, i.e., on prend en compte le fait que les instructions de chargement peuvent éventuellement engendrer des défauts de cache. Nous montrons comment traduire l'information dynamique sur la latence des instructions de chargement produite par DCPI (un outil de Compaq utilisant les compteurs matériels) en une information exploitable par l'ordonnanceur du compilateur Multiflow. Les gains obtenus sont modestes mais cette expérience nous permet de mieux comprendre les difficultés liées à l'exploitation d'informations dynamiques. [29]

7 Contrats industriels (nationaux, européens et internationaux)

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

- 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. Daniela Genius, de Philips, est collaborateur extérieur dans A3.

- 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; projet dans leur plate-forme (section 8.2.1) ;

– ST-MicroElectronics, en collaboration avec le projet CAPS de l'IRISA, dans le cadre du projet européen MEDEA/SMT (System level Methods and Tools), maintenant MEDEA+/MESA.

8 Actions régionales, nationales et internationales

8.1 Actions nationales

Olivier Temam est impliqué dans un contrat "Action Incitative Blanche" sur la simulation d'architectures (collaboration PRiSM, LRI (Orsay) et l'IRIT de Toulouse).

Thérèse Hardin, de Paris VI, est directeur officiel de la thèse de Ping Hu.

Véronique Donzeau-Gouge, professeur au CNAM est directeur officiel de la thèse de Pierre Amiranoff.

Pour le calcul des fenêtres généralisées (voir 6.2.1), nous collaborons avec Philippe Clauss de l'Université de Strasbourg. Guy-René Perrin 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. Les invités en 2000 ont été :

– Guang R. Gao, University of Delaware, Newark, Etats-Unis, 18 janvier 2000;

– Alexis Vartanian, LRI, Université de Paris-Sud, Orsay, 31 janvier 2000;

– Christèle Faure, avant-projet TROPICS, INRIA Sophia-Antipolis, 20 mars 2000;

– Toni Juan et Roger Espasa, Universitat Politecnica de Catalunya, Barcelona, Espagne, 24 mars 2000;

– Kathryn S McKinley, University of Massachusetts, Amherst, Etats-Unis, 5 mai 2000;

– Antonio Gonzalez, Universitat Politecnica de Catalunya, Barcelona, Espagne, 11 mai 2000;

– André Seznec, projet CAPS, IRISA, Rennes, 25 mai 2000;

8.2 Actions européennes

8.2.1 MHAOTEU

Participants : Aloke Bajpai, Cédric Bastoul, Christine Eisenbeis, Paul Feautrier, Abishek Prabhat, Andry Randrianatoavina, Olivier Temam, François Thomasset, Sid Ahmed Ali Touati, Grégory Watts.

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. Cette année, nous avons particulièrement travaillé sur l'analyse dynamique de la localité des données, par l'approche GRW [34] (sections 6.2.1 et 6.2.2), ainsi que sur la relation entre code source et code assembleur (section 6.1.4). Nous avons aussi étudié l'optimisation de code, par l'approche en phases, décrite dans la section 6.4.1. Ce projet se terminera en mars 2001. Dans ce cadre, Ch. Eisenbeis a passé une semaine à l'Université de Edinburgh (13 au 16 juin 2000) pour travailler sur la possibilité d'intégrer le pipeline logiciel dans l'environnement MARS, développé par Mike O'Boyle, basé sur les techniques polyédriques.

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 et avec Luc Bougé de l'ENSL (Ecole Normale Supérieure de Lyon) sur l'utilisation de Java pour le calcul hautes performances.

S. Lelait, ex-thésard, travaille maintenant chez Siemens (Münich) et continue à collaborer avec nous sur les structures du meeting graph [7], avec Dominique de Werra, de l'EPFL de Lausanne (Suisse) et Elena Stöhr, de l'Université de Manchester (Angleterre).

Sur la lancée du projet ESPRIT OCEANS, achevé fin 1999, nous travaillons toujours avec Mike O'Boyle, de l'Université d'Edinburgh et Peter Knijnenburg de l'Université de Leiden, Pays Bas. Dans ce cadre, nous avons accueilli en stage Arjan Salomons, de Leiden, pendant deux mois (du 1er mai au 30 juin 2000). Le but de son stage était la combinaison des techniques de pipeline logiciel avec l' "aplatissement de boucles" (*loop flattening*), destiné à favoriser la création de boucles à grand nombre d'itérations.

8.3 Actions internationales

O. Temam collabore régulièrement avec K. McKinley de l'Université du Massachussets.

Nous collaborons avec J.-L. Gaudiot de l'University of South California (Los Angeles) et G. Gao de l'Université de Delaware sur l'interaction entre parallélisme d'instructions et parallélisme de thread, dans le cadre d'un projet commun NSF/INRIA. Ch. Eisenbeis a passé une semaine (du 9 au 16 novembre 2000) à l'USC Los Angeles. Ping Hu a aussi passé une semaine dans l'équipe de Gao (Université de Delaware, Etats-Unis).

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 dans le laboratoire de David Padua. En retour, Peng Wu, doctorante du même laboratoire, a passé 2 mois au laboratoire PRiSM.

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 a fait partie du comité de programme de RenPar' 2000, CASES' 2000 et CC' 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".

Olivier Temam a été membre du comité de programme du Workshop MEDEA à PACT'2000, ainsi que de la conférence Sympa'7.

9.2 Enseignement universitaire

Pierre Amiranoff est 1/2 ATER à l'IIE-CNAM.

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

Jean-François Collard et Paul Feautrier enseignent la parallélisation automatique dans le cadre du DEA MISI (Université de Versailles Saint-Quentin) et du DEA SI (Université Pierre et Marie Curie).

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

Olivier Temam est depuis 1999 professeur à l'Université d'Orsay, où il donne le cours de DEA "Systèmes enfouis", le cours niveau Licence sur l'architecture de processeurs, et le cours niveau Licence sur les réseaux.

9.3 Participation à des colloques, séminaires, invitations

Les membres du projet ont donné les séminaires suivants :

- Séminaire du CSAR (Université de l'Illinois à Urbana Champaign) (Albert Cohen, "Program analysis and transformation : from the polytope model to formal languages"), 27 janvier 2000. Séminaire également donné au laboratoire IBM Watson, New York.
- Rencontre « HiPerf », École Nationale Supérieure des Mines de Paris (Albert Cohen, "Pointer Analysis for Monotonic Container Traversals", et Paul Feautrier "Analyse de références en Java"), 30 novembre 2000.
- Séminaire à l'USC (University of South California, Los Angeles), 16 novembre 2000 (Christine Eisenbeis: "Le projet OCEANS").
- Workshop "Java and High Performance Computing", Dagstuhl, Allemagne, 22 août 2000 (Paul Feautrier: "Java Reference Analysis"), séminaire donné aussi dans les circonstances suivantes:
  - HP Lab et Laboratoire de Compilation de ST-Microelectronics, Cambridge, Mass., 3 Novembre 2000 ;

  - Séminaire Compilation de l'UIUC, 14 Novembre 2000.

- Séminaire au laboratoire ST MicroElectronics/HP Labs (Cambridge) (Ping Hu, "Analysis and Optimization of Predicated Code").

- Fifth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR2000), 25-27 mai 2000, Rochester, NY, USA, (Ping Hu: "Static Analysis for Guarded Code")

- Séminaire au laboratoire CAPSL (University of Delaware) (Ping Hu: "Analysis of predicated code")

- IMACS WORLD CONGRESS 2000, Lausanne, Suisse, Août 2000 (Olivier Temam: "MHAOTEU Tools for Memory Hierarchy Management").

- Journée "Simulation et Petaflops", CEA, septembre 2000 (Olivier Temam: "Adéquation entre programmes et architectures")

- Workshop Interact-4 (Sid Ahmed Ali Touati: "Load-Store optimization for Software Pipelining").

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, décembre 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, juillet 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, janvier 1996.

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

Thèses et habilitations à diriger des recherches

[11] M. Dai, *Transformation et optimisation des programmes sources pour le pipeline logiciel*, thèse de doctorat, Université de Versailles–Saint-Quentin, 9 juin 2000.

[12] P. Hu, *Code Gardé : Traduction, Analyse Statique, Pipeline Logiciel*, thèse de doctorat, Université de Paris VI, 6 juillet 2000.

Articles et chapitres de livre

[13] W. Amme, P. Braun, F. Thomasset, E. Zehendner, «Data Dependence Analysis of Assembly Code», *International Journal of Parallel Programming 28*, 5, 2000, p. 431–467.

[14] D. Barthou, A. Cohen, J.-F. Collard, «Maximal Static Expansion», *Int. Journal of Parallel Programming 28*, 3, juin 2000, p. 213–243.

[15] P. Feautrier, «Les Compilateurs», *Technique et science informatiques 19*, 1-2-3, 2000, p. 223–232.

[16] M. Griebl, P. Feautrier, C. Lengauer, « On Index Set Splitting», *International Journal of Parallel Programming 28*, 4, 2000, p. 607–731.

[17] K. McKinley, O. Temam, « Quantifying loop nest locality using SPEC'95 and the perfect benchmarks», *ACM Transactions on Computer Systems 17*, 2000, p. 288–336.

[18] X. Redon, P. Feautrier, «Detection of Scans in the Polytope Model», *Parallel Algorithms and Applications 15*, 3-4, 2000, p. 229–263.

[19] E. Rohou, F. Bodin, C. Eisenbeis, A. Seznec, «Handling Global Constraints in Compiler Strategy», *International Journal of Parallel Programming 28*, 4, 2000, p. 325–345.

Communications à des congrès, colloques, etc.

[20] J. Abella, S. A. A. Touati, A. Anderson, C. Ciuraneta, J. M. Codina, M. Dai, C. Eisenbeis, G. Fursin, A. Gonzalez, J. Llosa, M. O'Boyle, A. Randrianatoavina, J. Sanchez, O. Temam, «MHAOTEU Tools for Memory Hierarchy Management», *in: 16th IMACS WORLD CONGRESS 2000, on Scientific Computation, Applied Mathematics and Simulation*, Lausanne, Switzerland, August 21-25 2000.

[21] P. Braun, W. Amme, F. Thomasset, E. Zehendner, «A data flow framework for analyzing assembly code », *in: Proc. of the 8th Workshop on Compilers for Parallel Computers*, École Normale Supérieure de Lyon (ENS Lyon), France, Aussois, France, janvier 2000.

[22] A. Cohen, « Program analysis and transformation: beyond the polytope model », *in : Proc. of the 8th Workshop on Compilers for Parallel Computers*, École Normale Supérieure de Lyon (ENS Lyon), France, Aussois, France, janvier 2000. Conférencier invité, extrait de la thèse.

[23] M. Dai, C. Eisenbeis, S.-A.-A. Touati, «Load-Store Optimization For Software Pipelining», *in : INTERACT-4 workshop at the 6th international Symposium on High-Performance Computer Architecture (HPCA)*, ACM SIGARCH Computer Architecture News, Toulouse, France, janvier 2000.

[24] I. Djelic, « Elimination de redondances pour architectures EPIC », *in : Symposium sur les Architectures Nouvelles de Machines (Sympa'6)*, Besançon, France, june 2000.

[25] N. Drach, J. Sebot, « SIMD ISA Extensions: Tradeoff between Power Consumption and Performance on a Superscalar Processor », *in : Kool Chips Workshop, Micro 33*, Monterey, California, dec 2000.

[26] A. Farcy, O. Temam, « Prédiction de branchement: les limites de la corrélation », *in : 6ème Symposiun sur les Architectures Nouvelles de Machines (SympA'6)*, p. 67–76, Besançon, june 2000.

[27] P. Hu, «Static Analysis for Guarded Code», *in : Proceedings of the Fifth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers(LCR2000) and LNCS 1915*, May 2000.

[28] J. Knoop, J.-F. Collard, R. Ju, « Partial Redundancy Elimination on Predicated Code », *in : Static Analysis Symposium (SAS'00)*, Santa Barbara, Californie, june 2000.

[29] G. Lindenmaier, K. S. McKinley, O. Temam, «Load Scheduling using Hardware Counters», *in : Euro-Par 2000*, Munich, Germany, Aug 2000.

Rapports de recherche et publications internes

[30] P. Amiranoff, «Optimisation de programmes annotés par des assertions», *rapport de recherche n° RR-3983*, Inria, août 2000.

[31] P. Feautrier, « Automatic Distribution of Data and Computations », *rapport de recherche n° 2000-3*, PRiSM, 2000.

[32] S.-A.-A. Touati, F. Thomasset, «Register Saturation in Data Dependence Graphs», *Rapport de recherche n° RR-3978*, INRIA, juillet 2000, `http://www.inria.fr/rrrt/rr-3978.html`.

[33] S.-A.-A. Touati, « Optimal Register Saturation in Superscalar and VLIW Codes », *Rapport de recherche*, INRIA, octobre 2000, `ftp://ftp.inria.fr/INRIA/Projects/a3/touati/optiRS.ps.gz`.

Divers

[34] A. Bajpai, «Dynamic GRW Computation and Visualisation Tool», juin 2000, Rapport de stage.

[35] D. de Werra, C. Eisenbeis, S. Lelait, E. Stöhr, « Circular arc graph coloring: on chords and circuits in the meeting graph», à paraître dans l'European Journal of Operations Research.

[36] A. Prabhat, «Group Based Mapping of Source to Assembly», juin 2000, Rapport de stage.

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

[38] S.-A.-A. TOUATI, « Register Saturation in Superscalar and VLIW Codes », Springer Verlag, Lecture Notes in Computer Science Series, Genova, Italy, *in: Proceedings of the International Conference on Compiler Construction*, avril 2001, à paraître.