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

François Thomasset [DR]

Personnel CNRS

Jean-François Collard [CR]

Personnel Université

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

Paul Feautrier [Professeur, Université de Versailles-Saint-Quentin]

Olivier Temam [Maître de conférences, Université de Versailles-Saint-Quentin, puis professeur, Université d'Orsay depuis le 1er septembre 1999]

Doctorants

Pierre Amiranoff [1/2 ATER à l'IIE/CNAM depuis le 1er septembre 1999]

Albert Cohen [AMN, Université de Versailles-Saint-Quentin]

Min Dai [bourse INRIA jusqu'au 30 avril 1999, Université de Versailles–Saint-Quentin]

Ivan Djelic [bourse MENRT, Université de Paris VI]

Alexandre Farcy [bourse MENRT, Université de Paris VI]

Ping Hu [bourse INRIA, Université de Paris VI]

Andry Randrianatoavina [bourse MENRT, Université de Strasbourg]

Sid Ahmed Ali Touati [bourse CROUS France/Algérie]

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

Stagiaires

Pierre Amiranoff [stage de DEA, CNAM, du 8 avril au 31 août 1999]

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 hiéarchie 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

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.

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].

L'analyse géométrique [4] 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 [8] l'algorithme

[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.

[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.

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 [7]. Le problème se 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 8.2.2) 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 (section 8.2.1) s'articule 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.

[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.

5 Logiciels

5.1 PiLo : pipeline logiciel

PiLo est un package de pipeline logiciel interfaçable, développé par Antoine Sawaya dans sa thèse [7]. 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 [8]), améliorée dans [7]. Il est utilisé dans l'environnement Sage++ (voir section 6.3.4) ainsi que dans l'environnement SALTO (voir section 8.2.1).

5.2 LoRA : allocation de registres dans les boucles

LoRA [26] est un package d'allocation de registres dans les boucles, développé par Sylvain Lelait dans sa thèse [5]. 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 [34]. 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

Participant : Paul Feautrier.

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

On sait l'importance que prend actuellement le langage JAVA pour toutes les applications liées à l'Internet. On sait également que des efforts sont faits pour que JAVA devienne un langage de programmation universel. L'obstable principal est que les nombreuses vérifications imposées par la norme ainsi que l'utilisation d'une machine virtuelle limitent très sévèrement les performances des applications JAVA.

Ces contraintes peuvent être levées moyennant une analyse des références, allant au-delà des classiques analyses de forme des structures de données (arbre, DAG, structures cycliques). Il s'agit ici d'analyser la forme des tableaux, et nous avons pour cela introduit la notion de "peigne" : un tableau dont chaque élément donne accès à une structure de données isolée. Il est facile de voir qu'un tableau en forme de peigne peut être traité comme un tableau FORTRAN, par exemple pour le calcul des dépendances.

Les analyses de pointeurs classiques, qui traitent un tableau comme un tout, ne sont pas assez précises pour nos besoins. Nous avons du mettre au point une combinaison d'analyse par instance et d'analyse par élément de tableau qui se montre d'une précision suffisante dans la majorité des cas.

Un de nos objectifs maintenant est de voir s'il est possible d'augmenter encore la précision de l'analyse en passant à un modèle totalement "par instance". Il faudra d'autre part examiner l'impact de nos analyses sur les compilateurs JAVA "Hautes Performances". Sont-elles suffisantes, et, sinon, dans quelles directions faut-il les approfondir ?

6.1.2 Analyse statique pour code gardé

Participant : Ping Hu.

Nous avons d'abord proposé un mécanisme pour déduire directement les sémantiques des gardes dans le contexte de code gardé. A partir des relations sémantiques entre les gardes, nous avons étendu systématiquement les définitions conventionnelles concernant les analyses de programmes prenant en compte les gardes dans les codes. De plus, nous avons développé des techniques d'analyse *gard-aware*, comprenant l'analyse de flot de contrôle, l'analyse de dépendances ainsi que l'analyse de flot de données. L'analyse *gard-aware* de flot de contrôle peut nous rendre les résultats traditionnels, tel que les dominances, les post-dominances et les équivalences de flot de contrôle, etc. L'analyse *gard-aware* de dépendances peut éliminer plusieurs types de dépendances afin d'offrir plus de chances pour détecter et extraire le parallélisme en présence de code gardé. Par ailleurs, les équations *gard-aware* de flot de données formeront une base essentielle pour développer les techniques *gard-aware* d'allocation de registres. En plus, nous avons détaillé un complet algorithme d'*if-conversion*. Cet algorithme a été amélioré sur

la base de notre analyse *gard-aware* d'équivalence de flot de contrôle afin de réduire le nombre de gardes dont on a besoin.

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 [23] 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 définitions visibles

Participant : Jean-François Collard.

Ce travail, en collaboration avec l'Université de Passau en Allemagne, cherche à approfondir nos résultats précédents et à rapprocher leurs formalismes de ceux plus généralement utlisés. Ce rapprochement nous a permi de montrer les gains de précisions que permettent nos techniques [13] et d'améliorer la robustesse de nos techniques [20]

6.1.5 Forme SSA pour programmes parallèles

Participant : Jean-François Collard.

La forme SSA est très utilisée dans les compilateurs de langages séquentiels manipulant des scalaires. Elle a été récemment étendue au cas des tableaux, extension indispensable à l'adoption de la SSA dans l'optimisation des codes de calcul scientifiques. Restait à élargir cette forme SSA pour tableaux aux programmes parallèles, ce qui a été fait dans [21].

6.1.6 Spécifier les programmes et leurs propriétés

Participants : Pierre Amiranoff, François Thomasset.

Certains codes "scientifiques" élaborés comme ceux utilisés par le projet Gamma dans les constructions de maillage pour les "éléments finis" sont irréductibles aux techniques classiques d'analyse de boucles. Le stage avait pour objet de débroussailler le terrain des spécifications formelles pour des programmes de maillage. En effet, les techniques de spécification formelle ont la particularité de dégager les propriétés mathématiques du programme: il est tentant de mettre cette faculté au service de l'optimisation du programme. L'objectif du travail [31] a été de développer avec le langage B et l'Atelier B un algorithme simple relatif au domaine du maillage (construction de l'enveloppe convexe d'un nuage de points dans le plan).

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 8.2.2).

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* dûs aux conflits mémoire sur un même *cache set* sont les plus difficiles à contrôler. Récemment, 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, 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 [3], 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 [28], en particulier le calcul de la taille de la projection d'un polyèdre paramétré. Nous étudions ces problèmes en partenariat avec l'Université de Strasbourg.

6.2.2 Fenêtres de référence généralisées - approche dynamique

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

Pour étudier la pertinence des GRW (fenêtres de référence généralisées), en dépit de la complexité de l'approche statique, nous avons développé l'outil DGRW d'évaluation dynamique

des fenêtres. Le but était de vérifier expérimentalement la coorélation entre la taille de la fenêtre et le nombre de *cache miss* dûs aux interférences. Dans notre cas, nous savons que:

$$\|GRW\| \leq 1 \Leftrightarrow nombre\_de\_conflict\_miss = 0$$

Cette propriété nous a permis de déceler dans le code Osiris (développé dans le projet ESTIME), des défauts de cache dûs aux interférences, et de vérifier qu'ils disparaissaient par transformation adéquate du code ("unroll-and-jam") [29].

6.2.3 Prédiction de borne de performance pour le parallélisme d'instruction et la hiérarchie mémoire

Participant : Sid Ahmed Ali Touati.

Ce travail est destiné à comprendre les performances d'un code sur une architecture à parallélisme d'instructions et hiérarchie mémoire. En général, les performances globales d'un code semblent peu satisfaisantes comparées aux capacités des machines. La question est de savoir si ce problème est dû à une mauvaise utilisation des ressources du processeur ou s'il est intrinsèque au programme considéré. Dans un processus d'optimisation, aussi bien automatique que manuel, les questions qui se posent sont : est-ce que le code est bien optimisé? Ne peut-on pas optimiser davantage? Faut-t-il arrêter là ou bien chercher encore à améliorer la performance?

Cette année, nous avons utilisé une approche basée sur l'analyse statique du code.Nos travaux dans ce sens sont une extension des travaux de Callahan et Kennedy[CCK88]. Ces derniers ont proposé un modèle mathématique pour borner le nombre de *stalls* dans le pipeline d'un processeur. Ils formulent une borne de performance d'une boucle en se basant sur les capacités de la machine d'une part, et le nombre d'opérations d'autre part. Cependant, leurs travaux se situent dans le cas des processeurs uni-instruction et ne prennent pas en compte les contraintes de registres, ni l'influence de la hiérarchie mémoire. Nous avons étendu leur modèle dans le cas des processeurs ILP (Instruction Level Parallelism), en prenant en compte les contraintes de registre et la hiérarchie mémoire :

1. pour le cas des processeurs ILP, nous classons les opérations de la boucle selon les ressources matérielles qu'elle consomment; ainsi nous pouvons traiter chaque ressource séparément comme dans le cas d'un processeur uni-instruction;

2. pour les contraintes des registres, nous utilisons la méthode URSA (Unified ReSsource Allocation for register and fonctional units in VLIW architectures [BGSL93]) pour la spécification des contraintes des registres. Cette méthode permet de modifier le graphe des dépendances de données du corps d'une boucle pour vérifier les contraintes de ressources

[CCK88] D. Callahan, J. Cocke, K. Kennedy, « Estimating Interlock and Improving Balance for Pipelined Architectures», *Journal of Parallel and Distributed Computing 5*, 4, août 1988, p. 334–358.

[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`.

et des registres de la machine. Nous utilisons l'extension de cette technique dans le cas de la boucle complète, développée dans le projet[San96].

3. enfin, des difficultés apparaissent lors de la spécification des latences des opérations mémoire. En effet, selon l'ordre d'exécution de ces opérations, la latence varie selon que l'accès est effectué dans tel ou tel niveau de la hiérarchie mémoire (cache L1, cache L2, mémoire). Pour cela, nous utilisons les modèles de prédiction qui ont été déjà étudiés l'an passé et éliminons les opérations mémoire redondantes, grâce à l'outil TOPS (voir 5.3). Cette simplification nous permet de nous concentrer sur la localité spatiale des accès mémoire, et d'utiliser des modèle probabilistes pour prédire la latence de ces opérations.

Pour mener à bien cette étude, nous avons développé quelques outils. "instop" est une première version d'évaluation dynamique de la performance optimiste. "timop" mesure le temps d'exécution réel d'une boucle en nombre de cycles. Il est utilisé pour comparer la performance réelle d'une boucle et la performance optimiste calculée par l'outil "instop". "visudep"permet de calculer les dépendances de données (flow, anti, output) d'une boucle, et l'affiche dans une interface conviviale: une fenêtre X avec des couleurs différentes pour chaque type de dépendance, ainsi que des formes différentes pour chaque type d'opération. C'est un outil pratique qui permet de déduire par exemple si la boucle est parallèle, ou si telle variable scalaire peut être un sujet à une opération de réduction ou de privatisation. Il peut être utilisé aussi pour analyser la structure de la boucle et décider d'appliquer une technique d'optimisation adéquate.

6.2.4 Propriétés de localité dans les programmes

Participant : Olivier Temam.

Nous avons utilisé l'analyse systématique des applications pour en identifier les principales caractéristiques afin de mieux focaliser les optimisations architecturales ou logicielles. Dans ce cadre, nous avons effectué une première étude sur les propriétés de localité des benchmarks Perfect Club. Cependant, ces benchmarks comportant des jeux de données de petite taille qui exercent insuffisamment la hiérarchie mémoire, nous avons ensuite effectué une étude similaire pour les benchmarks Spec95. De manière assez surprenante, la plupart des conclusions obtenues pour les programmes Perfect sont maintenues pour les programmes Spec. Ainsi, nous avons montré que plusieurs lieux communs sur les propriétés de localité des applications étaient incorrects et nous avons également mis en évidence plusieurs propriétés pouvant influer sur le développement futur d'applications. Par exemple, la plupart des optimisations pour la localité sont développées pour un nid de boucles mais nous avons montré qu'il était nettement plus important de travailler sur les défauts de cache *entre* boucles [39]. Cette étude est réalisée en collaboration avec Kathryn McKinley de l'université du Massachussetts.

[San96] J. Santini, *Ordonnancement et Allocation de Registres: Extendion de la Technique URSA aux boucles*, Rapport de DEA, Université d'Orléans, septembre 1996.

6.3 Optimisation des performances

6.3.1 Elimination des redondances partielles

Participants : Jean-François Collard, Ivan Djelic.

L'élimination des redondances partielles est une optimisation très courante dans les compilateurs de recherche et commerciaux, et 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 documents préliminaires sur le sujet ont été rédigés et sont soumis à publication [27, 25].

6.3.2 Allocation des registres dans les boucles

Participant : Christine Eisenbeis.

Nos études passées sur l'allocation des registres dans les boucles, basées sur le meeting graph, ont donné lieu à des recherches plus théoriques [36, 14], où l'on étudie les relations entre le degré chromatique du graphe d'interférences et le nombre de circuits dans le meeting graph [5].

6.3.3 Découpage de l'espace d'itérations

Participant : Paul Feautrier.

En collaboration avec M. Griebl et C. Lengauer (Université de Passau, Allemagne).

Les méthodes modernes de parallélisation basée sur le modèle polyédrique sont très puissantes, mais il a été montré que dans certains cas elles ne trouvent pas tout le parallélisme disponible. La raison est probablement que la représentation du parallélisme utilisée (les ordonnancements affines) n'ont pas assez de souplesse pour couvrir tous les cas possibles. Notre proposition a été de passer à des ordonnancements affines par morceaux. Le problème est alors de définir les morceaux. Ceci fait, les méthodes d'ordonnancement classiques s'appliquent sans changement.

Le principe de découpage que nous proposons utilise les dépendances, vues comme des relations entre espaces d'itérations. On sépare l'image de la dépendance et les points de l'espace d'itération qui ne sont pas dans l'image.

A de multiples dépendances correspondent de multiples découpages, qui peuvent être propagés et combinés de diverses façons. Notre travail s'attache à identifier les découpages utiles, mais pour le moment les règles proposées ne sont que des heuristiques [22].

Ce travail sera poursuivi dans deux directions. D'une part il faut affiner les règles de découpage et identifier le plus vite possible les découpages utiles. Nous avons déjà obtenu des résultats partiels dans cette direction. On peut d'autre part penser à des découpages dynamiques à l'exécution ; tout reste à faire dans cette direction.

6.3.4 Pipeline dans le programme source et pré-chargement

Participants : Min Dai, Christine Eisenbeis, Sid Ahmed Ali Touati.

A l'origine, la technique de pipeline logiciel est utilisée dans l'optimisation de code assembleur, à bas niveau. Or, toutes les informations nécessaires à sa mise en œuvre sont présentes dans le code à haut niveau – indice de boucle, bornes des boucles, calcul des adresses des tableaux ...– et disparaissent dans le processus de génération de code. De plus, le développeur de code n'a aucun moyen de contrôler l'ordonnancement de son code, alors qu'il peut être détenteur d'informations primordiales, par exemple qu'une donnée n'a sûrement pas été utilisée avant et causera un défaut de cache lors de son chargement. C'est pourquoi nous avons développé l'outil TOPS (section 6.3.4), de restructuration de code par pipeline logiciel, dans l'environnement de parallélisation Sage++ (développé à l'Université d'Indiana). La boucle FORTRAN est tout d'abord découpée en code à 3 adresses. Les variables temporaires sont stockées dans des tableaux (un tableau par variable du corps de la boucle, chaque élément de tableau correspond à une itération). Le graphe de dépendances est ensuite déterminé, puis envoyé à un logiciel de pipeline logiciel, qui renvoie les dates de lancement des différentes opérations. Les informations liées à l'architecture peuvent aussi être spécifiées par l'utilisateur. Pour provoquer le pré-chargement des données, il peut par exemple indiquer une grande latence d'accès à la mémoire. L'allocateur de registres alloue ensuite les tableaux temporaires dans des variables scalaires, qui seront a priori sauvegardées en registres par tous les compilateurs. Ainsi, l'utilisateur a le contrôle sur l'ordonnancement de son code à bas niveau et peut mettre au point les performances de son code de manière fine.

Cette année, nous avons commencé à étudier expérimentalement l'influence de la valeur donné à la latence (grande pour provoquer un pré-chargement) sur les performances du code, avec des résultats encourageants [30]. Nous intégrons de plus dans le processus d'optimisation une phase d'élimination des chargements mémoire redondants. Eliminer les chargements mémoire redondants diminue la pression sur les accès mémoire qui sont souvent le facteur bloquant de la performance. En retour, on accroît ainsi la pression sur les registres et on peut générer du *spill code* qui aurait été inutile dans le code original. Le choix des accès mémoire à éliminer est basé sur les circuits critiques du graphe de dépendances. [34, 35].

6.3.5 Modélisation des contraintes d'architecture

Participant : Christine Eisenbeis.

Dans le cadre du projet OCEANS (section 8.2.1), nous avons travaillé sur la modélisation de l'architecture TriMedia de Philips par tables de réservation. Cette modélisation était auparavant basée sur la notion de slots d'occupation et nécessitait la remise en cause de l'allocation dans les slots à chaque essai d'ordonnancement d'une opération dans une combinaison déjà formée d'instructions parallèles. Notre nouvelle modélisation permet de décider directement si la nouvelle combinaison est valide ou non [24, 17].

7 Contrats industriels (nationaux, européens et internationaux)

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

- Fujitsu, dans le cadre d'un contrat avec le FECIT (Toulouse), où nous étudions les processeurs de Fujitsu [37];
- Hewlett-Packard ("Performance Delivery Laboratory" de Cupertino, USA et Bristol, Angleterre), pour analyser les techniques mises en œuvre dans les compilateurs industriels.
- Philips, Pays-Bas est partenaire du projet ESPRIT OCEANS, dans lequel nous optimisons du code pour leur processeur TriMedia (section 8.2.1) ;
- l'ONERA, France est partenaire dans le projet ESPRIT MHAOTEU ; ils nous fournissent des programmes à optimiser sur les architectures COMPAQ/Digital-alpha (section 8.2.2) ;
- Edinburgh Portable Compilers, Royaume-Uni est partenaire dans MHAOTEU, pour intégrer les résultats de ce projet dans leur plate-forme (section 8.2.2) ;
- ST-MicroElectronics, en collaboration avec le projet CAPS de l'IRISA, dans le cadre du projet européen MEDEA/SMT (System level Methods and Tools).

8 Actions régionales, nationales et internationales

8.1 Actions nationales

Olivier Temam collabore avec Jean-Luc Béchennce du LRI (Orsay) sur la création d'un environnement de simulation d'architecture (ASF). Il est aussi 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 co-encadre la thèse de Pierre Amiranoff avec François Thomasset.

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.

Paul Feautrier est membre du bureau de l'ORAP (Organisation Associative du Parallélisme). A ce titre, il organise ou participe à l'organisation d'un Forum annuel où sont présentés à une audience de non spécialistes les derniers développements du calcul parallèle.

A3 organise un séminaire commun avec le groupe CRI (Centre de Recherches en Informatique) de l'École des Mines de Paris. Les invités en 1999 ont été :

- Francky Catthoor et Thierry Omnès (IMEC, Louvain), 25 janvier 1999;
- Daniela Genius, Université de Karlsruhe, Allemagne, 8 février 1999;
- R. Govindarajan, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore, Inde, 14 juin 1999;

- Sylvain Lelait, Université technologique de Vienne, Autriche, 14 juin 1999;
- David Gregg, Université technologique de Vienne, Autriche, 15 octobre 1999;
- Guillaume Huard, Ecole Normale Supérieure de Lyon, 15 octobre 1999;
- Jean-Claude Sogno, INRIA Rocquencourt, 19 novembre 1999;
- Stefano Crespi-Reghizzi, Politecnico de Milan, Italie, 6 décembre 1999;
- Jean-Louis Giavitto, LRI, Orsay, 14 décembre 1999;
- Eberhardt Zehendner, Université de Iéna, Allemagne, 14 décembre 1999.

8.2 Actions européennes

8.2.1 OCEANS

Participants : Christine Eisenbeis, Ping Hu.

OCEANS (Optimizing Compilers for Embedded ApplicatioNS) est un projet ESPRIT LTR [16] regroupant comme site académiques l'Université de Leiden (Pays-Bas), l'Université de Manchester (Royaume-Uni), l'Université de Versailles-Saint-Quentin et l'INRIA (responsable François Bodin, IRISA). OCEANS s'est terminé fin octobre 1999. Il s'agissait d'appliquer nos techniques bien aguerries de parallélisation des boucles et d'ordonnancement d'instructions aux processeurs VLIW récents, et en particulier au processeur VLIW TriMedia de Philips (Pays-Bas) qui est aussi partenaire dans le consortium. Dans ce cadre, PiLo (voir 5.1) et LoRA (voir 5.2) ont été intégrés dans le logiciel SALTO (System for Assembly Language Tools and Optimisation) développé dans l'équipe CAPS à l'IRISA. L'approche adoptée dans le projet était basée sur une compilation itérative, par combinaison de trasnformations dans le code source et dans le code assembleur [33]. Dans le cadre d'OCEANS, nous avons aussi développé une nouvelle méthode [24, 17], voir section 6.3.5, de modélisation de l'architecture du processeur TriMedia, qui est désormais utilisée chez Philips.

8.2.2 MHAOTEU

Participants : Christine Eisenbeis, Andry Randrianatoavina, Olivier Temam, François Thomasset, Sid Ahmed Ali Touati, Gregory 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 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 statique et dynamique de la localité des données, par l'approche GRW [28] (sections 6.2.1 et 6.2.2), ainsi que sur une approche logicielle de pré-chargement des données [30] (section 6.3.4). Nous avons aussi étudié l'optimisation du code Osiris développé dans le projet ESTIME [29].

8.2.3 Autres collaborations

Dans le cadre de 2 contrats PROCOPE (France-Allemagne), nous collaborons avec Eberhard Zehendner, Peter Braun et Wolfram Amme de l'Université de Iena sur l'analyse de code assembleur (section 6.1.3), et avec l'équipe de Christian Lengauer, de l'Université de Passau sur l'analyse de flot de données pour structures irrégulières. Dans le cadre du premier contrat, nous avons accueilli cette année Eberhard Zehendner et Wolfram Amme (19 au 23 juillet et 12 au 16 décembre). Nous leur avons aussi rendu visite (10 au 14 mai et 21 au 26 novembre).

S. Lelait, ex-thésard, est depuis septembre 1996 à l'Université Technologique de Vienne (Autriche) et continue à collaborer avec nous sur les structures du meeting graph [5], avec Dominique de Werra, de l'EPFL de Lausanne (Suisse) et Elena Stöhr, de l'Université de Manchester (Angleterre). Nous avons aussi accueilli David Gregg, de cette même équipe, du 1er au 30 octobre 1999 (travail sur le pipeline des boucles avec branchement).

Nous coopérons également avec Jens Knoop de l'Université de Dortmunt, spécialiste des analyses des définitions visibles basées des techniques de point-fixe. Nos propres techniques étant basées sur la programmation linéaire en nombres entiers, cette coopération permet de compléter notre approche par une autre théorie et une autre technologie.

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.

Dans le cadre d'un accord CNRS-UIUC, Paul Feautrier a passé 3 mois à l'université de Urbana-Champaign (Illinois), dans le laboratoire de David Padua.

J.-F. Collard a monté un projet commun franco-russe de l'Institut Liapunov avec A. Lastovetsky (ISPRAS, Moscou), sur la compilation de langages data-parallèles sur architectures à mémoire partagée, comme les réseaux de stations de travail (intégration du paralléliseur automatique `PAF` de Versailles et du compilateur `mpC` de l'ISPRAS.

Nous collaborons aussi avec R.Govindarajan, du SERC (Supercomputer Education and Research Center) de l'IISc (Indian Institute of Science), de Bangalore (Inde), sur le pipeline logiciel en présence de spéculation sur les données. Il a passé deux mois (mai et juin 1999) dans notre équipe.

Ping Hu a passé sept mois dans le laboratoire ChiPTec a l'Université d'Adelaide en Australie, où elle travaillait avec Michael Liebelt, sur l'analyse de code gardé.

Nous collaborons aussi avec Zaher Mahjoub, de la Faculté des Sciences de Tunis (Tunisie), dans le cadre d'une action bilatérale).

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 Vincent Messé (Université de Rennes, 30 mars 1999). Elle a fait partie du comité de programme de RenPar' 11, ICS' 99 et CC' 2000.

Paul Feautrier fait partie du comité de rédaction de "IEEE Transactions on Parallel and Distributed Systems", "International Journal of Parallel Programming" et de " Parallel Computing". Il est membre de l'Organisation Associative du Parallélisme (ORAP) depuis sa fondation, et aussi membre du Conseil Scientifique du CNUS/CINESC depuis cette année.

François Thomasset a participé au comité de rédaction du numéro spécial (Janvier 2000) de TSI.

9.2 Enseignement universitaire

Pierre Amiranoff est 1/2 ATER à l'IIE-CNAM. Olivier Temam est depuis 1999 professeur à l'Université d'Orsay. François Thomasset assurait jusqu'en Mars 1999 des cours du DEA LIFO (analyse statique et génération de code) à l'Université d'Orléans. Sid Ahmed Ali Touati était chargé de cours à l'Université de Créteil, il est maintenant chargé de TD/TP à l'Université de Versailles-Saint-Quentin.

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

9.3 Participation à des colloques, séminaires, invitations

Les membres du projet ont donné les séminaires suivants :

- Workshop on Code Transformations for Data Transfer and Storage Efficient Execution of Multimedia Applications on (Parallel) Processors, 15 septembre 1999, IMEC, Louvain, Belgique (Min Dai: "Elimination of Redundant LoadStore").

- "Fine-grain Parallelism", Université d'Adelaïde, Australie (Ping Hu);

- Second International Workshop on Compiler and Architecture Support for Embedded Systems (CASES' 99), 1-3 octobre 1999, (Christine Eisenbeis, Intervention d'Erven Rohou: "Flexible issue slot assignment for VLIW architectures" et Jean-François Collard: "Partial Redundancy Elimination on Predicated Code: An Extension");

- Séminaire "Instruction-Level Parallelism and Parallelizing Compilation", 18-20 avril 1999, Dagstuhl, Allemagne, (Jean-François Collard: "Array-Tailored Analysis vv. MFP Analysis", Christine Eisenbeis: "OCEANS; Optimizing Compilers for Embedded Applications").

- Séminaire du CSAR (Université de l'Illinois à Urbana Champaign) (Paul Feautrier, "Compiling for Massively Parallel Architectures : A Perspective"), 24 Mars 1999, conférence également donnée au «Kolloquium» de l'Université de Passau, 22 Juin 1999.

- Séminaire de la Chaire d'informatique de l'Université de Passau (Paul Feautrier, «Smart Memories, an Introduction»).
- Colloque «Compilers for Parallel Architectures», Mont Saint-Odile, (Paul Feautrier, "Index Set Splitting", 19 Octobre 1999), séminaire également donné aux Journées Parallélisme'99, Faculté des Sciences de Tunis, 24 Novembre 1999.
- DEA d'Informatique de la Faculté des Sciences de Tunis (Tutoriel de Paul Feautrier, "Génération de Code Parallèle", 25Novembre 1999).

10 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] 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.

[2] 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.

[3] 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.

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

[5] S. Lelait, *Contribution à l'allocation de registres dans les boucles*, Thèse de Doctorat, Université d'Orléans, janvier 1996.

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

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

[8] 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

[9] A. Cohen, *Program Analysis and Transformation: from the Polytope Model to Formal Languages*, thèse de doctorat, Université de Versailles-Saint-Quentin, 21 décembre 1999.

[10] J.-F. Collard, *Contribution en analyse et optimisation de programmes pour compilateurs parallélisants*, habilitation à diriger des recherches, Université de Versailles-Saint-Quentin, 7 décembre 1999.

[11] O. Temam, *Pour une meilleure adéquation entre programmes et architectures de processeurs*, habilitation à diriger des recherches, Université de Versailles-Saint-Quentin, 8 janvier 1999.

Articles et chapitres de livre

[12] A. Cohen, « Analyse de flot de données pour programmes récursifs à l'aide de langages algébriques», *Technique et science informatiques 18*, 3, 1999, p. 323–343.

[13] J.-F. Collard, «Analyse des définitions visibles : État de l'art et applications en parallélisation automatique», *Technique et science informatiques 18*, 4, avril 1999, p. 373–395.

[14] D. de Werra, C. Eisenbeis, S. Lelait, B. Marmol, « On a graph-theoretical model for cyclic register allocation», *Discrete Applied Mathematics 93*, 2-3, July 1999, p. 191–203.

[15] O. Temam, « An Algorithm for Optimally Exploiting Spatial and Temporal Locality in Upper Memory Levels », *IEEE Transactions on Computers, special issue on Cache Memories 48*, 2, février 1999.

Communications à des congrès, colloques, etc.

[16] M. Barreteau, F. Bodin, Z. Chamski, H. Charles, C. Eisenbeis, J. Gurd, J. Hoogerbrugge, P. Hu, W. Jalby, T. Kisuki, P. Knijnenburg, P. van der Mark, A. Nisbet, M. O'Boyle, E. Rohou, A. Seznec, E. Stohr, M. Treffers, H. Wijshoff, « OCEANS: Optimising Compilers for Embedded Applications», *in : EuroPar'99 Parallel Processing*, *Lecture Notes in Computer Science*, 1685, Springer-Verlag, p. pp. 1171–1175, Toulouse, août 1999.

[17] Z. Chamski, C. Eisenbeis, E. Rohou, « Flexible Issue Slot Assignment for VLIW Architectures», *in : Proceedings of the Second International Workshop on Compiler and Architecture Support for Embedded Systems (CASES'99)*, Washington, D.C., 1-3 octobre 1999. résumé de [24], `http://www.capsl.udel.edu/conferences/cases99/papers/paper14.ps`.

[18] A. Cohen, V. Lefebvre, « Optimization of Storage Mappings for Parallel Programs », *in : EuroPar'99*, *LNCS*, Springer-Verlag, p. 375–382, Toulouse, France, septembre 1999.

[19] A. Cohen, « Parallelization via Constrained Storage Mapping Optimization », *in : Int. Symp. on High Performance Computing (ISHPC'99)*, *LNCS*, 1615, Springer-Verlag, p. 83–94, Kyoto, Japan, mai 1999.

[20] J.-F. Collard, M. Griebl, « A Precise Fixpoint Reaching Definition Analysis for Arrays », *in : Proc. Workshop on Languages and Compilers for Parallel Computing*, *LNCS*, Springer-Verlag, San Diego, CA, août 1999.

[21] J. Collard, «Array SSA for Explicitly Parallel Programs», *in : Euro-Par'99*, *LNCS*, Springer-Verlag, Toulouse, France, août 1999.

[22] M. Griebl, P. Feautrier, C. Lengauer, «On Index Set Splitting», *in : PACT'99*, 1999.

Rapports de recherche et publications internes

[23] W. Amme, P. Braun, E. Zehendner, F. Thomasset, « Data Dependence Analysis of Assembly Code », *Rapport de Recherche n° RR-3764*, INRIA, Rocquencourt, Septembre 1999, `http://www.inria.fr/RRRT/RR-3764.html`.

[24] Z. Chamski, C. Eisenbeis, E. Rohou, « Flexible Issue Slot Assignment for VLIW Architectures », *rapport de recherche n° 3784*, INRIA, octobre 1999, version étendue de [24], `http://www.inria.fr/RRRT/RR-3784.html`.

[25] J.-F. Collard, «Partial Redundancy Elimination on Predicated Code: an Extension», *rapport de recherche*, PRiSM, 1999.

[26] 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`.

[27] J. Knoop, J.-F. Collard, R. Ju, «Partial Redundancy Elimination on Predicated Code», *Rapport interne*, PRiSM, 1999.

Divers

[28] J. Abella, N. Bermudo, C. Ciuraneta, J. M. Codina, C. Eisenbeis, A. Gonzalez, J. Llosa, A. Randrianatoavina, F. Thomasset, S. A. A. Touati, X. Vera, «Advanced data locality analysis», Deliverable M2.D1 of the MHAOTEU ESPRIT project no 24942, décembre 1999.

[29] J. Abella, M. Bull, C. Ciuraneta, J. M. Codina, C. Eisenbeis, P. Guillen, A. Gonzalez, J. Llosa, M. O. Boyle, A. Randrianatoavina, O. Temam, F. Thomasset, S. A. A. Touati, X. Vera, G. Watts, «Process for Optimizing an Application», Deliverable M2.D3 of the MHAOTEU ESPRIT project no 24942, décembre 1999.

[30] J. Abella, J. M. Codina, C. Ciuraneta, M. Dai, C. Eisenbeis, A. Gonzalez, J. Llosa, P. Knijnenburg, M. O'Boyle, S. A. A. Touati, X. Vera, «Data Prefetching and Targeted Loop Optimizations», Deliverable M2.D2 of the MHAOTEU ESPRIT project no 24942, décembre 1999.

[31] P. Amiranoff, *Spécification de problèmes de maillage*, Rapport de DEA, CNAM, septembre 1999.

[32] D. Barthou, A. Cohen, J.-F. Collard, «Maximal Static Expansion», à paraître dans l'Int. Journal of Parallel Programming.

[33] F. Bodin, Z. Chamski, C. Eisenbeis, S. Karapatis, T. Kisuki, P. Knijnenburg, P. van der Mark, N. Motogelwa, M. O'Boyle, E. Rohou, «Iterative Compilation: Validation Report», Deliverable 3.3 of the OCEANS ESPRIT project no 22729, octobre 1999.

[34] M. Dai, « Transformations de programmes pour le parallélisme d'instructions », Thèse en préparation.

[35] M. Dai, C. Eisenbeis, S. A. A. Touati, «Load-store Elimination for Software Pipelining», soumis.

[36] 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.

[37] C. Eisenbeis, A. Lichnewsky, «Etude du VPP Fujitsu», Rapport de contrat, décembre 1999.

[38] P. Feautrier, «Les Compilateurs», à paraître dans Techniques et Sciences Informatiques.

[39] K. Mckinley, O. Temam, « Quantifying Loop Nest Locality Using SPEC'95 and the Perfect Benchmarks», à paraître dans ACM Transactions on Computer Systems.

[40] X. Redon, P. Feautrier, « Detection of Scans in the Polytope Model », à paraître dans le Journal of Parallel Algorithms and Applications.