INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet A3

Analyse Avancée Appliquée à l'optimisation des codes

Rocquencourt

THÈME 1A

Table des matières

1. Composition de l'équipe

Responsable scientifique
Christine Eisenbeis [CR Inria, DR Inria depuis le 1er septembre 2002]

Assistante de projet
Nathalie Gaudechoux [SAR Inria]

Personnel Inria
Albert Cohen [CR]
Paul Feautrier [Professeur de l'Université de Versailles-Saint-Quentin, en délégation jusqu'au 31 août 2002, puis professeur de l'Ecole Normale Supérieure de Lyon à partir du 1er septembre 2002]
François Thomasset [DR]
Philippe Clauss [Professeur de l'Université de Strasbourg, en délégation à partir du 1er septembre 2002]

Ingénieur associé
Saurabh Sharma [à partir du 1er octobre 2002]

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

Collaborateur extérieur
Daniela Genius [Bourse Marie Curie, Philips Research France, jusqu'au 31 août 2002]

Doctorants
Christophe Alias [bourse MENRT, Université de Versailles-Saint-Quentin]
Pierre Amiranoff [professeur certifié de mathématiques, détaché comme 1/2 ATER CNAM-Paris]
Cédric Bastoul [bourse MENRT, Université de Paris VI]
Ivan Djelic [bourse MENRT, 1/2 ATER, Université de Versailles St-Quentin]
Sid Ahmed Ali Touati [bourse INRIA, jusqu'au 30 juin 2002]

Stagiaires
Patrick Carribault [Maîtrise de l' Université de Versailles-Saint-Quentin]
Messaoud Kara [DEA de l' Université de Versailles-Saint-Quentin, du 1er avril au 31 août 2002]
Sebastian Pop [DESS de l'Université de Strasbourg, du 1er juin au 31 août 2002]
Olivier Virloulet [Maîtrise de l' Université de Versailles-Saint-Quentin]

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

3.1. Introduction

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 ».

3.2. 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 [38], cherchent à associer une propriété à chaque point du programme [42]. 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 [33]. 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 [32].

L'analyse géométrique [8] 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.

3.3. 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 [46], 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 [12] 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 [41][37][40], voir aussi [11]. Le problème se réduit alors à un problème de modélisation des contraintes.

4. Domaines d'application

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 (*Digital Signal Processor*, processeur de traitement du signal), mais aussi les serveurs de calcul à petit nombre de processeurs, ou encore les supercalculateurs présentant un modèle de programmation à mémoire partagée.

[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.

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, 1998-2001), les applications de type multimédia (projet OCEANS, 1996-1999), ou vidéographiques (projet SANDRA (section 7.1)).

L'objectif de performance peut se décliner sous différents aspects : maximisation de la vitesse d'exécution, minimisation de transferts ou d'accès mémoire, minimisation de la consommation ou de la puissance dissipée par un processeur, minimisation de la taille du code généré, minimisation du coût de conception du processeur.

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 (1998-2001) visait ce type d'applications pour l'optimisation de la hiérarchie mémoire.
- pré-processeur d'optimisation source à source : c'est le cas de PAF (Parallèliseur Automatique de FORTRAN) ou de TOPS (pipeline logiciel source à source, section 5.8).
- compilateur ;
- post-processeur d'optimisation assembleur vers assembleur, comme dans la plate-forme SALTO du projet CAPS de l'IRISA, auxquels nos logiciels PILO et LORA sont intégrés ;
- de plus, les travaux de A3 permettent de caractériser la difficulté d'utilisation de chaque dispositif des processeurs spécialisés, et donc d'influer sur leur architecture, comme dans le cas du projet SANDRA.

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

5. Logiciels

5.1. DiST : un outil pour la simulation distribuée de processeurs

DIST, un outil pour la simulation distribuée de processeurs. Afin de raccourcir les cycles de simulation/optimisation en recherche en architecture ou en compilation, DiST accélère la simulation d'un benchmark isolé en découpant sa trace d'exécution en morceaux. La précision des résultats est garantie par un mécanisme de recouvrement dynamique entre morceaux. L'outil est conçu de telle sorte que l'adaptation des simulateurs existants soit la plus simple possible (parfois moins de 10 lignes de code).

5.2. Tuareg : un environnement très complet pour la programmation en Objective Caml sous Emacs

TUAREG, un environnement très complet pour la programmation en Objective Caml sous Emacs. Distribué sous licence GPL, fourni avec Debian GNU/Linux 3.0. Développement depuis janvier 1997. Des milliers d'installations dans des dizaines de pays, universités, laboratoires de recherche (sur AFS à Rocquencourt), lycées (classes préparatoires) et entreprises (Thalès, France-Télécom, EDF, Xerox).

5.3. DigLC2 : un simulateur « au niveau des portes logiques » pour le processeur LC-2

Participants : Albert Cohen, Olivier Temam.

DIGLC2 est un simulateur « au niveau des portes logiques » pour le processeur LC-2 (*Little Computer 2* de Yale Patt et Sanjay Patel) [19]. Dédié à l'enseignement de l'architecture des ordinateurs et à l'introduction aux bases de la compilation et du système, il est distribué sous licence GPL, sous forme de circuit pour le simulateur DigLog de Chipmunk (également GPL).

5.4. CLooG : Code de parcours des points entiers d'un polyèdre

Participant : Cédric Bastoul.

CLooG (*Chunky LOOp Generator*) est un logiciel et une bibliothèque générant des boucles pour parcourir les points entiers d'un polyèdre http://www.prism.uvsq.fr/~cedb/bastools/cloog.html. CLooG trouve le code ou le pseudo-code où chaque point d'un ensemble de polyèdres paramétrés est atteint. Bien que l'utilisateur ait le contrôle total sur la qualité et la forme du code généré, CLooG a été conçu pour éviter la surcharge de contrôle et produire un code particulièrement efficace. Le cœur du processus de génération de code [22] est l'algorithme de Quilleré et al. [45]. Cette technique génère les niveaux de boucles en projetant les polyèdres sur la dimension correspondante, puis en séparant les projetés en polyèdres disjoints. Elle trie ensuite ces polyèdres de manière à respecter l'ordre lexicographique, et génère enfin récursivement les nids de boucles qui parcourent chaque polyèdre.

Cet outil a été conçu pour s'intégrer en phase finale des optimiseurs de code source à source travaillant dans le domaine polyédrique tels que certains paralléliseurs automatiques (PAF) ou optimiseurs de gestion des caches (CHUNKY). Il est particulièrement orienté « code compilable » et permet d'effectuer simplement des transformations de programmes.

Diverses améliorations et extensions sont envisagées :

- support des sauts de boucles différents de l'unité,
- support des synchronisations pour la génération de code parallèle,
- automatisation du compromis taille de code/efficacité.

5.5. PipLib

Participants : Paul Feautrier, Cédric Bastoul.

PIP est un solveur déjà bien connu de problèmes paramétrés de programmation linéaire en nombres entiers http://www.prism.uvsq.fr/~cedb/bastools/piplib.html. PIP est un logiciel qui trouve le minimum lexicographique de l'ensemble des points entiers d'un polyèdre convexe, même si ce polyèdre dépend linéairement d'un ou plusieurs paramètres. L'utilisateur peut demander une solution non entière : PIP donnera alors la solution exacte sous la forme d'un quotient d'entiers. Le cœur de PIP est l'algorithme paramétrique des coupes de Gomory suivi par la méthode paramétrique du simplexe dual [7][23]. La bibliothèque PIPLIB a été implémentée pour permettre à l'utilisateur d'utiliser PIP directement depuis ses programmes, sans appels système ou accès à des fichiers. L'utilisateur doit juste pouvoir lier son programme à des bibliothèques C.

5.6. PiLo : pipeline logiciel

PILO est un package de pipeline logiciel interfaçable, développé par Antoine Sawaya dans sa thèse [11]. 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 [12]), améliorée dans [11]. Il est utilisé dans l'environnement Sage++ (transformation source à source) ainsi que dans l'environnement SALTO (projet ESPRIT OCEANS). La nouvelle version de SALTO développée dans le projet CAPS de l'IRISA s'appelle ALISEE . PILO doit être connecté à ALISEE , ainsi que dans le projet SANDRA au travers d'une interface XML.

5.7. LoRA : allocation de registres dans les boucles

LORA [6] est un package d'allocation de registres dans les boucles, développé par Sylvain Lelait dans sa thèse [9]. LORA est basé sur le *meeting graph* (voir section 5.7). 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).

5.8. TOPS : pipeline logiciel source à source

TOPS est un outil permettant de pipeliner les boucles dans les programmes source (FORTRAN). L'utilisateur ou le compilateur désigne les boucles à pipeliner en insérant des directives spéciales, et peut aussi spécifier, après analyse ou par expérience personnelle, quelles données devraient être chargées en avance, pour éviter de bloquer le processeur en cas de défaut de cache. TOPS a été développé par Min Dai dans sa thèse [34]. Il utilise l'environnement Sage++ de manipulation de programmes.

5.9. SAREQ : test d'équivalence de systèmes d'équations de récurrence

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

6. Résultats nouveaux

6.1. Analyse sémantique des programmes

6.1.1. Paradigme d'analyse de programmes récursifs par instances

Participants : Pierre Amiranoff, Albert Cohen, Paul Feautrier.

Nous avons poursuivi [28] la formalisation et l'extension de l'analyse par instances pour programmes récursifs, avec la définition d'un langage dédié à la détermination statique des variables inductives. Ce langage impose des contraintes sur le modèle de programmation et met en évidence les variables et les structures de données mises en œuvre dans l'analyse. Il se nomme MOGUL, pour *MOnoid GUided Language*, l'ensemble des propriétés du flot de contrôle et du flot de données étant exprimées dans des monoïdes de type fini (a priori quelconques). Dans le cadre des programmes récursifs opérant sur des tableaux, nous avons construit un modèle hybride issu de la théorie des langages formels et d'une sous-classe décidable de machines de Minsky. Ce modèle permet de définir un test de dépendances aussi précis mais plus général que celui étudié dans la thèse de A. Cohen [2]. Pour le langage MOGUL, une implémentation complète (du parsing à la construction des transducteurs rationnels décrivant les dépendances) a été réalisée en OCaml. Nous avons montré la NP-complétude de ce test, mais nous proposons un algorithme efficace en pratique sur les programmes étudiés (plusieurs centaines de lignes de code MOGUL), fondé sur un encodage en programmation linéaire en nombres entiers.

6.1.2. Analyse de code à prédicats

Participants : Albert Cohen, Ivan Djelic, Sebastian Pop.

L'élimination des redondances partielles, la propagation de constantes, l'allocation de registres et l'ordonnancement des instructions constituent autant d'optimisations cruciales pour les architectures récentes. Ces techniques sont bien maîtrisées, mais un nouveau mécanisme architectural – présent notamment

sur l'IA 64 (Itanium) – remet en cause les conditions d'application et les performances des algorithmes existants : il s'agit de la « prédication », c'est-à-dire l'exécution conditionnelle des instructions. Dans le cadre de l'élimination de redondances partielles, un article est paru en 2002 dans un journal [15]. L'algorithme d'analyse de prédicats a été revu en profondeur, avec preuve complète dans le formalisme de l'interprétation abstraite [32] (sémantique de traces assez originale) et extension à tous les types d'instructions manipulant les prédicats. Cet algorithme permet de généraliser très simplement n'importe quelle analyse de flot de données de propriétés booléennes sur des graphes de flot de contrôle quelconques (*bit-vector analysis*) ; nous l'avons entre autres appliqué à l'allocation de registres. L'application à l'élimination de redondances partielles à également été prouvée (un treillis complexe mais peu coûteux a dû être construit) et remotivée avec des considérations plus théoriques et des hypothèses plus fortes sur la forme de la représentation intermédiaire du compilateur. Les deux algorithmes ont été implémentés dans la partie terminale (*code generator*) du compilateur libre Open64/ORC (ORC est coordonné par Intel et l'Académie des Sciences de Chine, Open64 est l'ancien compilateur de SGI), et l'implémentation est en cours de stabilisation. La prochaine étape (fin de la thèse d'Ivan Djelic) consiste à appliquer systématiquement ces algorithmes sur des benchmarks réels, pour vérifier leur passage à l'échelle en pratique puis obtenir des gains de performances.

6.2. Optimisation des performances des programmes

6.2.1. Modèle et environnement pour l'optimisation semi-automatique

Participants : Cédric Bastoul, Albert Cohen, Sylvain Girbal, Saurabh Sharma, Olivier Temam.

Nous développons un modèle unifié intégrant des analyses et des transformations de code polyédriques originales dans un compilateur moderne (Open64/ORC, Cray/SGI et Intel). Ce modèle utilise des phases d'analyse dynamique de programmes (simulation) et une méthodologie d'optimisation, pour piloter les analyses statiques et les transformations. Nous avons conçu une méthode de simulation distribuée permettant d'accélérer l'analyse dynamique d'un benchmark unique sur un réseau de stations. Cette méthode a été implémentée dans un outil distribué librement [39] et appliquée à 3 simulateurs existants (SimpleScalar 3 et SimAlpha [35], d'une part, et un simulateur fonctionnel de PowerPC G3 en Système C [43], d'autre part). Un article est soumis à une conférence. Le modèle unifié est en cours de formalisation ; nous tentons également de combiner plusieurs transformations classiques (fusion de boucles, partitionnement, substitution, pipeline logiciel) dans une même transformation, afin de réduire l'espace des paramètres d'optimisation. Le travail dans le compilateur ORC est actuellement effectué par Saurabh Sharma ; il consiste à réaliser une interface de conversion de la représentation intermédiaire WHIRL de haut niveau vers une description polyédrique des nids de boucles, et vice-versa (la conversion inverse s'appuie sur le générateur de code CLooG de C. Bastoul).

Nous envisageons à terme la construction d'un « centre d'optimisation de programmes » pour valoriser nos outils et notre expertise dans le domaine de l'optimisation pour architectures à hautes performances et embarquées.

6.2.2. Amélioration de la localité des programmes réguliers

Participants : Cédric Bastoul, Paul Feautrier.

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

Dans notre approche, nous proposons de « court-circuiter » le mécanisme de remplacement, dont le fonctionnement est très difficile à prévoir. Le programme est divisé en morceaux (*chunks*). Au début de chaque morceau, le cache est vide. La taille du morceau est ajustée de façon que ses données tiennent juste dans le cache. Le mécanisme de remplacement n'intervient donc pas. A la fin du morceau, le cache est vidé et on passe au morceau suivant.

Dans ce modèle, il est possible de donner une estimation asymptotique du trafic et d'utiliser cette information pour calculer un découpage optimum. Le découpage peut donner lieu, le cas échéant, à des restructurations du programme source. Dans le cadre de son travail de thèse, C. Bastoul a défini les algorithmes de découpage pour les cas les plus importants : réutilisation de groupe, auto-réutilisation simple et multiple et le prototype CHUNKY a été réalisé. Le programme est ensuite reconstruit à l'aide du générateur de code CLOOG(section 5.4). Les premières campagnes de test se sont révélées particulièrement concluantes [17].

Dans la version actuelle, la localité spatiale, la localité temporelle propre et de groupe sont prises en compte, mais aucun tuilage n'est encore effectué. La génération de code produit quant à elle un code très efficace mais qui peut saturer le cache d'instructions. L'exploitation du tuilage et la production de code exploitant le cache d'instructions feront l'objet de la troisième année de thèse de C. Bastoul.

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

6.2.3. Allocation cyclique de registres dans les boucles non ordonnancées

Participants : Sid-Ahmed-Ali Touati, Christine Eisenbeis.

L'allocation de registres dans les boucles internes est généralement effectuée pendant ou après une phase de pipeline logiciel. Ceci car une allocation classique de registres avant la phase d'ordonnancement ne peut être efficacement effectuée à cause du manque d'information sur les interférences entre les durées de vie des variables. Ainsi, nous risquerions d'introduire un grand nombre de fausses dépendances à cause de la réutilisation des registres. Ces fausses dépendances, si elles ne respectent pas la phase de pipeline logiciel qui viendrait par la suite, risquent de heurter considérablement le parallélisme d'instructions.

Pour éviter ce problème, nous travaillons seulement sur le graphe de dépendances de données et contrôlons le besoin en registres (*register pressure*) en ajoutant des flèches de précédence artificielles, dites flèches de réutilisation. En restreignant l'espace mémoire (ici le nombre R de registres disponibles) utilisé, nous restreignons bien sûr le parallélisme, qui est mesuré dans le cas du pipeline logiciel par le facteur MII (*Minimum Initiation Interval*). Notre modélisation utilise cependant R et MII en paramètres, ce qui nous permet de faire un compromis entre les deux. Elle consiste en deux temps ; choix des flèches de réutilisation, choix des distances de réutilisation sur les flèches. Nous donnons une formulation par programme linéaire en nombres entiers de notre modélisation. Une réutilisation hamiltonienne nous permet de modéliser l'utilisation des registres à décalage (*rotating register files*) présents par exemple dans l'Itanium. Si nous fixons à l'avance les flèches de réutilisation, le problème de minimisation de R à MII fixé est plus facile (matrice « presque » uni-modulaire) mais nous ne savons pas s'il est encore NP-complet ou polynomial. Ce dernier cas généralise un résultat précédent de Ning et Gao [44] où seules les auto-réutilisations étaient prises en compte [27][13][31].

En collaboration avec William Jalby et Christophe Lemuet de l' Université de Versailles-Saint-Quentin, nous avons réussi avec succès à porter notre technique sur l'architecture EPIC/IA64 [30]. Dans le futur, notre approche sera étendue pour prendre en compte d'autres méthodes efficaces d'ordonnancement d'instructions, comme la vectorisation des accès mémoire, afin de ne pas se restreindre au pipeline logiciel. Nos résultats préliminaires sont très encourageants [30].

6.2.4. Inclusion automatique d'objets en Java

Participants : Patrick Carribault, Albert Cohen, Olivier Virlouvet.

Nous avons poursuivi l'exploration de la notion d'*inclusion d'objets* (*object inlining*, *unboxing*) pour Java. Notre approche s'appuie sur la séparation entre objets dynamiques et variables, sur le *ramasse-miettes* automatique (*garbage collector*), et sur la présence de classes de *conteneurs* de haut niveau. Nous avons formalisé la sémantique de l'inclusion d'objets, sur un sous-ensemble de Java, en étendant le modèle introduit par Julian Dolby [36]. En pratique, la détection de l'inclusivité des objets est donc réalisée à la compilation, mais les contraintes liées au bytecode Java requièrent le report de l'optimisation proprement dite à la

génération de code natif par le JIT (compilateur *Just In Time*). L'implémentation précédente, réalisée par Patrick Carribault, visait la machine virtuelle Kaffe. Celle-ci est trop primitive pour obtenir des comparaisons de performances valables, nous avons donc développé un nouveau modèle d'annotations du bytecode Java et une nouvelle implémentation dans l'environnement multi-JIT multi-GC (*Garbage Collector*) ORP (Open Runtime Platform, licence GPL, coordonné par Intel). Le développement est rendu beaucoup plus ardu, étant donné la masse et la complexité de l'environnement ORP. Un article est en cours de préparation, les premières expériences étant satisfaisantes.

6.3. Réseaux de processus : modélisation,ordonnancement

6.3.1. Prise en compte des ressources dans l'ordonnancement

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

Cette étude entre dans le cadre du projet SANDRA (section 7.1). Un réseau de processus de Kahn (KPN) est composé d'un ensemble fini de processus qui communiquent entre eux par des canaux FIFO disposant d'une capacité de stockage infinie. Chaque processus exécute un programme séquentiel pouvant utiliser des instructions de lecture (bloquante) ou d'écriture (non bloquante) sur un canal spécifié. De plus, un KPN doit respecter la règle suivante : un canal ne peut être connecté qu'à un seul processus en entrée et à un seul processus en sortie. Si le comportement des processus est déterministe, alors G. Kahn a montré que l'histoire de chaque canal est également déterministe. L'intérêt des KPN en synthèse de systèmes embarqués est qu'ils permettent de décrire des systèmes réactifs comportant du parallélisme, tout en se prêtant à une analyse « par dépendances » analogue à celle qui permet de traiter les codes séquentiels. Ils conduisent d'autre part à une représentation graphique qui est familière aux spécialistes du traitement du signal et de l'électronique.

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

Le prototype d'ordonnanceur YAKA, dont l'implémentation a débuté l'an dernier a été étendu de plusieurs façons :

- YAKA dispose désormais de deux *front-ends*, l'un pour un langage dérivé de C, l'autre pour une représentation intermédiaire présentée en XML. Cette dernière représentation est peu lisible, mais elle permet d'ajouter à peu de frais des informations qu'il serait délicat d'incorporer à la syntaxe de C (par exemple des informations temporelles).
- nous avons défini une heuristique pour traiter les contraintes de ressources, en les simulant au moyen de dépendances de données (voir aussi la section 6.3.3). Des résultats intéressants ont déjà été obtenus, mais la méthode montre une certain instabilité. Pour y remédier, il nous faut modifier la fonction objectif de notre optimiseur. Ce travail est en cours.
- YAKA s'appuie sur des algorithmes de calcul formel, écrits à l'origine en Maple. Les programmes ont maintenant été traduits en MuPAD à l'aide du logiciel de traduction développé pour l'occasion par F. Thomasset en Ocaml (voir http://www-rocq.inria.fr/~thomasse/Maple-MuPad/). Dans ce cadre, F. Thomasset a aussi développé une interface directe entre MuPAD et le logiciel Pip (Parametric Integer Programming).

6.3.2. Modélisation et vérification d'applications de traitement vidéo

Participants : Albert Cohen, Daniela Genius.

Dans le cadre du projet SANDRA (section 7.1), après avoir identifié un domaine d'applications spécifiques (flux vidéo, 3D, temps réel, régularité et contrôle statique), nous avons poursuivi le développement du modèle de *réseaux hiérarchiques de processus* ou *hierarchical process networks* (HPN). Ce modèle permet de traiter un ensemble restreint d'applications de traitement du signal en temps réel, mais il décrit très

précisément l'exécution et les ressources (unités fonctionnelles, mémoire, bande passante) nécessaires. Les deux caractéristiques principales des HPN sont les suivantes : d'une part une description explicite de la structure hiérarchique des données (pixels, tuiles, lignes, etc.) et des évènements associés, d'autre part un modèle d'activation périodique « relâché » (avec rafales et variations) mais avec des bornes maximales garanties. Le modèle a pu être validé sur quelques algorithmes de traitement vidéo et a donné lieu à une publication en conférence [18]. Malgré plusieurs extensions significatives, le modèle théorique peine à prendre en compte certains algorithmes partiellement irréguliers liés au graphisme 3D et des problèmes de précision ne sont pas encore résolus. Nous étudions également les interactions avec l'ordonnancement, la description de la machine et l'allocation de ressources (pipeline logiciel hiérarchique), en collaboration avec Ch. Eisenbeis et P. Feautrier, ainsi que Marc Duranton et Zbigniew Chamski de Philips Research.

6.3.3. Nids de boucle, pipeline logiciel, ressources

Participants : Christine Eisenbeis, Paul Feautrier, Messaoud Kara.

Dans ce travail, nous avons cherché à combiner deux approches, l'ordonnancement affine des nids de boucle (que nous appelons YAKA en référence avec notre logiciel qui implémente cette méthode, voir 6.3.1) et l'ordonnancement des boucles simples par le pipeline logiciel (dite PILO). La seconde approche a l'avantage de traiter des contraintes de ressources très fines et son but est de maximiser le débit de la boucle. La première approche cherche au contraire à minimiser la latence du programme (sa durée d'exécution) hors toute contrainte de ressource.

La question centrale est donc de savoir comment introduire des contraintes de ressource dans YAKA. Dans son stage de DEA [26], Messaoud Kara a tenté plusieurs approches sur divers exemples. On peut d'abord utiliser YAKA seul en affectant des ressources a priori aux opérations. Cette approche a l'inconvénient de restreindre trop tôt l'espace des solutions et est forcément sous-optimale. On peut ensuite tenter de généraliser PILO en le combinant avec des transformations simples de boucles telles le *unroll-and-jam*, qui permet d'exécuter dans une même itération des opérations correspondant à des itérations externes dans le code original. Cette approche reste cependant locale aux boucles les plus internes. La dernière approche est alors de combiner YAKA et PILO, le but étant que YAKA génère de « bonnes » boucles pour PILO. On se rend compte très vite que YAKA doit travailler sur du code de haut niveau, c'est-à-dire sans découper les opérations en opérations élémentaires à trois adresses (chargement depuis la mémoire, calcul, rangement du résultat en mémoire). Ensuite, il est primordial que le parallélisme généré par YAKA soit borné pour que PILO gère ensuite finement les contraintes de ressources et que le code résultant soit régulier.

Nous étudions désormais quelles contraintes de dépendances produisent un parallélisme borné ; il semble que ce soient les contraintes simulant justement les transformations de *unroll-and-jam*, l'étape suivante consistera alors à trouver quelles contraintes, ajoutées artificiellement, nous donneront le meilleur résultat pour un facteur de parallélisme fixé. Cette étude [25] entre aussi dans le cadre du projet SANDRA (section 7.1).

7. Contrats industriels

7.1. Projet Sandra

Le projet SANDRA est une collaboration entre l'INRIA, le PRF (Philips Research France) et le Philips NatLab (Eindhoven, Pays-Bas). Daniela Genius, qui bénéficie au PRF d'une bourse Marie Curie est collaborateur extérieur dans notre projet A3.

Le sujet de la collaboration est un environnement de définition et programmation pour piloter les processeurs graphiques et vidéo. L'avènement de la télévision numérique entraîne en effet la convergence des domaines de traitements graphiques et vidéo vers un cadre commun, où les flots d'images provenant de sources diverses (canaux de télévision, pages web, etc) seront intégrés en vue de leur restitution sur un récepteur à haute résolution. Les traitements consistent par exemple à changer l'échelle d'un flot d'images, ou à le composer avec des images graphiques ; ils sont soumis à de fortes contraintes de bande passante et de temps réel. Le problème posé concerne la construction d'environnements permettant la programmation de telles unités de

traitement, et si possible l'élaboration de compilateurs reciblables. Notre collaboration porte sur les points suivants :

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

Cette année, nous avons principalement travaillé sur les réseaux hiérarchiques de processus (section 6.3.2), les ordonnanceurs (sections 6.3.1 et 6.3.3), en parallèle avec la définition du langage d'entrée, de l'architecture, et de sa description.

8. Actions régionales, nationales et internationales

8.1. Actions nationales

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

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 exposés suivants ont été donnés en 2002 :

- 11 Février 2002 :*Probabilistic Data Flow Analysis and its Applications* par Bernhard Scholz (Inst. f. Computersprachen, TU Wien, Autriche) ;
- 11 Février 2002 : *Sémantiques probabilistes et analyse* par David Monniaux (Laboratoire d'informatique, École Normale Supérieure, Paris) ;
- 14 Février 2002 : *Memory Latency : to Tolerate or to Reduce ?* par Prof. Jean-Luc Gaudiot (University of California, Irvine, États-Unis) ;
- 14 Février 2002 : *Parallel Computation Methods for Genome-Level Bioinformatics* par Prof. Guang R. Gao (University of Delaware, Newark, États-Unis) ;
- 26 Mars 2002 : *Embedded System Design based on Transport Triggered Architectures* par Henk Corporaal (Delft University of Technology, Delft, Pays-Bas) ;
- 11 Avril 2002 : *Évolution de GCC/g++, optimisations, point de vue compilateur et programme utilisateur* par Gabriel Dos Reis (ENS de Cachan) ;
- 16 Mai 2002 : *SafeTSA : An Alternative to Stack-Based Mobile-Code Representations* par Wolfram Amme (Friedrich-Schiller-Universität, Jena, Allemagne) ;
- 16 Mai 2002 : *Propagation des Erreurs d'Arrondi dans les Calculs en Précision Finie* par Matthieu Martel (CEA/LIST-DTSI-SLA) ;
- 30 Mai 2002 : *Optimisation de code pour l'infographie* par Karine Brifault (PRiSM, Université de Versailles-Saint-Quentin) ;
- 7 Novembre 2002 : *Une méthode pour la recherche des comportements intelligents* par Eduardo Sanchez (Logic Systems Laboratory, EPFL, Lausanne, Suisse).

Ph. Clauss, A. Cohen, Ch. Eisenbeis et P. Feautrier font partie du comité de pilotage du réseau thématique pluridisciplinaire (RTP) du CNRS qui a été mis en place en septembre 2002, sur le thème « Architecture et compilation ». Ph. Clauss est co-responsable du RTP avec Pascal Sainrat de l'IRIT (Toulouse). Ph. Clauss et Ch. Eisenbeis animent conjointement une action spécifique (AS) sur le thème « Compilation pour les systèmes embarqués ». Les membres du projet participent aussi à l'action spécifique sur le thème « Nouvelles technologies et nouveaux paradigmes d'architectures », animé par Nathalie Drach-Temam du LRI.

8.2. Actions européennes

8.2.1. EUNECS

Olivier Temam fait partie du comité de pilotage de la proposition de réseau d'excellence EUNECS (European Network of Excellence on Computing Systems) sur l'interaction en compilation et architecture des processeurs. Les chercheurs de A3 sont membres de ce réseau.

8.2.2. Autres collaborations

Dans le cadre d'un contrat PROCOPE (France-Allemagne), nous collaborons avec l'équipe de Christian Lengauer, de l'Université de Passau. Deux sujets de recherches sont en cours : l'amélioration des techniques polyédriques d'ordonnancement (pavage, décomposition, ressources), et la reconnaissance d'algorithmes sur des structures de données avec pointeurs. Nous avons reçu le professeur Chris Lengauer en janvier 2002. P. Feautrier était à Passau du 9 au 22 juin 2002. A cette occasion, il a mis au point une méthode d'optimisation des communications qui permet ultérieurement un tuilage arbitraire du programme parallèle résultant [20].

8.3. Actions internationales

Au titre d'un contrat CMCU (Commission Mixte (franco-tunisienne) pour la Coopération Universitaire), nous collaborons avec l'équipe du professeur Zaher Mahjoub (Mohamed Jemni et Yosr Slama) de la Faculté des Sciences de Tunis.

Dans le cadre d'une collaboration NSF-INRIA entre l'Université de South California (USC) puis de l'Université de Irvine de Los Angeles, et l'Université de Delaware, nous avons reçu le professeur Jean-Luc Gaudiot (14-15 janvier, 15 avril, 4 octobre) et le professeur Gao (14-15 janvier). Nos recherches s'orientent vers l'optimisation des programmes pour une meilleure exploitation des « I-structures » dans les modèles data-flow.

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 des résultats

9.1. Animation de la communauté scientifique

A. Cohen faisait partie du jury de thèse de Julien Sébot (LRI, 8 novembre 2002). Il est le correspondant entre la direction et les projets de recherche de Rocquencourt dans la préparation du 6ème PCRD.

Ch. Eisenbeis était rapporteur de la thèse de Grégory Watts (LRI, 27 septembre 2002). Avec Tom Conte (NC State University, États-Unis) et Mary-Lou Soffa (University of Pittsburgh, États-Unis), elle organise en février 2003 un séminaire à Dagstuhl (Allemagne) sur le thème de l'optimisation logicielle des nouveaux processeurs.

P. Feautrier fait partie des comités de rédaction des journaux « International Journal of Parallel Programming », et « Parallel Computing ». Il était dans le jury d'habilitation de Pierre Boulet (2 décembre 2002, Lille) et les jurys de thèse de Olivier Glück (12 juillet 2002, Paris 6), Alexandre Farcy (12 septembre 2002, Paris 6) et Nga Nguyen (19 novembre 2002, Ecole des Mines de Paris).

9.2. Enseignement universitaire

C. Bastoul est moniteur de l'université Paris 6 et enseigne en TD/TP d'algorithmique et d'initiation à la programmation (langage PASCAL) en seconde année de DEUG SCM.

Ph. Clauss enseigne dans le DEA de l'Université Louis Pasteur de Strasbourg (cours « Parallélisme » et option « Compilation à hautes performances »).

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

Ch. Eisenbeis et F. Thomasset ont assuré le cours de compilation avancée au DEA de l'université d'Orléans.

P. Feautrier a enseigné dans les DEA du MISI (Université de Versailles-Saint-Quentin) et du SIR (Université Pierre et Marie Curie). Il a été nommé professeur à l'Ecole Normale Supérieure de Lyon au 1er septembre 2002.

9.3. Participation à des colloques, séminaires, invitations

Les membres du projet ont donné les séminaires et participé aux conférences suivants :

- Séminaire CompSys, INRIA Lyon (P. Feautrier, « Sur l'équivalence de deux systèmes d'équations de récurrence »), 6 février 2002.
- Séminaire au LIFO de l'Université d'Orléans (Pierre Amiranoff et A. Cohen, « De nouveaux résultats sur l'analyse statique de programmes récursifs »), mars 2002.
- Participation au Symposium « International Symposium on New Trends in Compiler Technology, Binary rewriting for faster and smaller programs, running everywhere », Ch. Eisenbeis and P. Feautrier, 7 mars 2002.XS
- Renpar'14, Rencontres du parallélisme, Hammamet, Tunisie (C. Bastoul, « Une méthode d'amélioration de la localité basée sur des estimations asymptotiques du trafic »), participation de P. Feautrier, 10-13 avril 2002.
- Ecole des Mines de Paris (C. Bastoul, « Transformations de programmes pour l'amélioration de la localité des données »), 30 avril 2002.
- Présentation au workshop « WCAE'02 » associé à la conférence « International Symposium on Computer Architecture » (ISCA'02), Anchorage, Alaska, États-Unis (A. Cohen, « Digital LC-2 : from bits & gates to a little computer »), participation à la conférence ISCA'02 et aux workshops associés, mai 2002.
- Laboratoire ASIM, université de Paris 6 (C. Bastoul, « Vers une technique de compilation pour les systèmes à mémoires locales »), 11 juillet 2002.
- Présentation à « Euro-Par », Paderborn, Allemagne (A. Cohen et Daniela Genius, « Multi-Periodic Process Networks », Denis Barthou, « On the Equivalence of Two Systems of Affine Recurrence Equations »), participation à la conférence et au tutoriel Itanium-2 associé, août 2002.
- Séminaire au laboratoire ICPS de l'Université Louis Pasteur de Strasbourg (A. Cohen, « De nouveaux résultats sur l'analyse statique de programmes récursifs » et « Multi-Periodic Process Networks »), septembre 2002.
- Participation aux conférences EMSOFT 2002, Grenoble (François Thomasset) et CASES 2002 (Ch. Eisenbeis et F. Thomasset), octobre 2002.
- Séminaire au CEA Saclay (A. Cohen, « De nouveaux résultats sur l'analyse statique de programmes récursifs » et « Pointer Analysis for Container-Centric Applications », et Sylvain Girbal « Unification de transformations polyédriques »), novembre 2002.
- Journées du Réseau Thématique Prioritaire SECC « Systèmes Embarqués Complexes ou Contraints » (Ph. Clauss : « Compilation pour les systèmes embarqués »), 14-15 novembre 2002, participation de Ch. Eisenbeis.
- Participation à la conférence « Micro-35 » et les workshops associés (notamment EPIC'2 et tutoriel ORC), Istanbul, Turquie (C. Bastoul, A. Cohen, Ivan Djelic, groupe de travail sur « l'analyse de code à prédicats » et sur l'optimisation semi-automatique, présentation aux principaux membres du projet Open64/ORC au Microprocessor Research Lab d'Intel), novembre 2002.
- Laboratoire PRiSM, université de Versailles (C. Bastoul, « Amélioration de la localité à la compilation : le chunking »), 18 novembre 2002.

10. Bibliographie

Bibliographie de référence

[1] P. CLAUSS. *Counting Solutions to Linear and Nonlinear Constraints through Ehrhart polynomials : Applications to Analyze and Transform Scientific Programs.* in « ACM Int. Conf. on Supercomputing », ACM, mai, 1996.

[2] 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.

[3] 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.

[4] C. EISENBEIS, W. JALBY, A. LICHNEWSKY. *Compiler techniques for optimizing memory and register usage on the CRAY2.* in « International Journal on High Speed Computing », numéro 2, volume 2, 1990, pages 193-222, http://www.inria.fr/rrrt/rr-1302.html, appeared also as INRIA Research Report no 1302 October 1990.

[5] C. EISENBEIS, W. JALBY, D. WINDHEISER, F. BODIN. *A Strategy for Array Management in Local Memory.* in « Mathematical Programming », volume 63, 1994, pages 331-370, Special Issue on Applications of Discrete Optimization in Computer Science.

[6] C. EISENBEIS, S. LELAIT. *LoRA : a Package for Loop Optimal Register Allocation.* Rapport de recherche, numéro 3709, INRIA, Rocquencourt, juin, 1999, http://www.inria.fr/rrrt/rr-3709.html.

[7] P. FEAUTRIER. *Parametric integer programming.* in « RAIRO Recherche Opérationnelle », numéro 3, volume 22, 1988, pages 243-268.

[8] P. FEAUTRIER. *Dataflow Analysis of Scalar and Array References.* in « Int. J. of Parallel Programming », numéro 1, volume 20, février, 1991, pages 23-53.

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

[10] K. S. MCKINLEY, O. TEMAM. *A Quantitative Analysis of Loop Nest Locality.* in « ASPLOS' 96 », Cambridge, Massachussets, octobre, 1996.

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

[12] J. WANG, C. EISENBEIS, M. JOURDAN, B. SU. *DEcomposed Software Pipelining : a New Perspective and a New Approach.* in « International Journal on Parallel Processing », numéro 3, volume 22, 1994, pages 357-379, Special Issue on Compilers and Architectures for Instruction Level Parallel Processing.

Thèses et habilitations à diriger des recherche

[13] S.-A.-A. TOUATI. *Register Pressure in Instruction Level Parallelism.* thèse de doctorat, Université de Versailles, France, juin, 2002, ftp://ftp.inria.fr/INRIA/Projects/a3/touati/thesis/.

Articles et chapitres de livre

[14] D. DE WERRA, C. EISENBEIS, S. LELAIT, E. STÖHR. *Circular arc graph coloring : on chords and circuits in the meeting graph.* in « European Journal of Operational Research », volume 136, 2002, pages 483-500.

[15] I. DJELIC. *Élimination de redondances partielles pour architectures EPIC.* in « Technique et science informatiques », 2002.

Communications à des congrès, colloques, etc.

[16] D. BARTHOU, P. FEAUTRIER, X. REDON. *On the Equivalence of Two Systems of Affine Recurrence Equations (Research Note).* in « Lecture Notes in Computer Science », série LNCS, volume 2400, pages 309-313, Paderborn, Allemagne, août, 2002, http://link.springer-ny.com/link/service/series/0558/bibs/2400/24000309.htm.

[17] C. BASTOUL. *Une méthode d'amélioration de la localité basée sur des estimations asymptotiques du trafic.* in « Renpar'14 », Hammamet, Tunisie, avril, 2002.

[18] A. COHEN, D. GENIUS, A. KORTEBI, Z. CHAMSKI, M. DURANTON, P. FEAUTRIER. *Multi-Periodic Process Networks : Prototyping and Verifying Stream-Processing Systems.* in « EuroPar'02 », série LNCS, volume 2400, Springer-Verlag, pages 137-146, Paderborn, Allemagne, août, 2002, http://link.springer-ny.com/link/service/series/0558/bibs/2400/24000137.htmf.

[19] A. COHEN, O. TEMAM. *Digital LC-2 : From Bits & Gates to a Little Computer.* in « Proc. of the 9th Workshop on Computer Architecture Education (WCAE'02) », Anchorage, Alaska, États-Unis, mai, 2002.

[20] M. GRIEBL, P. FEAUTRIER, A. GROESSLINGER. *Forward Communication Only Placements.* in « Workshop on Languages and Compilers for Parallel Computing », University of Maryland, États-Unis, juillet, 2002.

[21] P. WU, P. FEAUTRIER, D. PADUA, Z. SURA. *Instance-wise points-to analysis for loop-based dependence testing.* in « Proceedings of the 16th international conference on Supercomputing », ACM Press, pages 262-273, New York, New York, États-Unis, 2002, http://doi.acm.org/10.1145/514191.514228.

Rapports de recherche et publications internes

[22] C. BASTOUL. *Generating loops for scanning polyhedra.* rapport technique, numéro 2002/23, Université de Versailles-Saint-Quentin, 2002.

[23] P. FEAUTRIER, J. F. COLLARD, C. BASTOUL. *Solving systems of affine (in)equalities.* rapport technique, Université de Versailles-Saint-Quentin, 2002.

[24] D. GENIUS, A. COHEN, A. KORTEBI, P. FEAUTRIER, M. DURANTON, Z. CHAMSKI, C. EISENBEIS, L. PASQUIER, V. RIVIERRE, Q. ZHAO. *Hierarchical Process Networks for Prototyping and Verifying Stream*

Processing Systems. rapport de contrat, Philips, juillet, 2002.

[25] D. GENIUS, C. EISENBEIS, P. FEAUTRIER. *First experiments with the SANDRA scheduling tools.* rapport de contrat, Philips, juillet, 2002.

[26] M. KARA. *Etude expérimentale de l'ordonnancement sous contrainte de ressources.* Rapport de DEA, Université de Versailles-Saint-Quentin, septembre, 2002.

[27] S.-A.-A. TOUATI, C. EISENBEIS. *Cyclic Register Pressure and Aladdress for Modulo Scheduled Loops.* rapport technique, numéro 4442, INRIA, avril, 2002, http://www.inria.fr/rrrt/rr-4442.html.

Divers

[28] P. AMIRANOFF, A. COHEN, P. FEAUTRIER. *Instancewise Array Dependence Test for Recursive Programs.* in « Workshop on Compilers for Parallel Computers », Amsterdam, Pays-Bas, janvier, 2003.

[29] C. BASTOUL, P. FEAUTRIER. *Improving data locality by chunking.* in « Workshop on Compilers for Parallel Computers », Amsterdam, Pays-Bas, janvier, 2003.

[30] W. JALBY, C. LEMUET, S.-A.-A. TOUATI. *Efficient Code Optimization Technique for Itanium2 Cache System and Scientific Computing.* in « Workshop on Compilers for Parallel Computers », Amsterdam, Pays-Bas, janvier, 2003.

[31] S.-A.-A. TOUATI. *Minimizing the Register Requirement in Data Dependence Graphs.* in « Workshop on Compilers for Parallel Computers », Amsterdam, Pays-Bas, janvier, 2003.

Bibliographie générale

[32] 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 », pages 238-252, janvier, 1977.

[33] P. COUSOT, N. HALBWACHS. *Automatic Discovery of Linear Restraints Among Variables of a Program.* in « Fifth Annual ACM Symp. on Principles of Programming Languages », pages 84-97, Tucson, Ariz., janvier, 1978.

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

[35] R. DESIKAN, D. BURGER, S. W. KECKLER. *Measuring Experimental Error in Microprocessor Simulation.* in « The 28th Annual Intl. Symposium on Computer Architecture », pages 266-277, June, 2001.

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

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

[38] R. W. FLOYD. *Assigning Meaning to Programs.* in « Proc. of the Symp. in Applied Mathematics, Vol. 19 », AMS, éditeurs J.T. SCHWARTZ., pages 19-32, Providence, 1967.

[39] S. GIRBAL, G. MOUCHARD. *DiST.* 2002, http://www.microlib.org/DiST.

[40] 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.

[41] C. HANEN. *Study of a NP-hard cyclic scheduling problem : the recurrent job-shop.* in « European Journal of Operational Research », volume 72, January, 1994, pages 82-101.

[42] J. KAM, J. ULLMANN. *Global Data Flow Analysis and Iterative Algorithms.* in « Journal of the ACM », numéro 1, volume 23, janvier, 1976, pages 158-171.

[43] G. MOUCHARD. *PowerPC G3 simulator.* 2002, http://www.microlib.org.

[44] Q. NING, G. R. GAO. *A Novel Framework of Register Allocation for Software Pipelining.* in « Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages », pages 29-42, Charleston, South Carolina, 1993, http://citeseer.nj.nec.com/ning93novel.html.

[45] F. QUILLERÉ, S. RAJOPADHYE, D. WILDE. *Generation of efficient nested loops from polyhedra.* in « International Journal of Parallel Programming », numéro 5, volume 28, october, 2000, pages 469-498.

[46] B. R. RAU, C. D. GLAESER. *Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing.* in « Proceedings of the 14th Conference on Microprogramming and Microarchitecture », pages 183-198, octobre, 1981.