Avant-projet A3

Analyse Avancée de code Appliquée à l'optimisation

Localisation : *Rocquencourt*

Mots-clés : analyse statique de code, interprétation abstraite, parallélisme, hiérarchie mémoire, optimisation de code, allocation de registres, pipeline logiciel, analyse d'alias, pointeurs, ordonnancement d'instructions.

1 Composition de l'équipe

Responsable scientifique

Christine Eisenbeis, CR Inria

Responsable permanent

Alain Deutsch, CR Inria

Secrétaires

Josy Baron, Inria (Assistante commune au Bât. 08)
Muriel de Bianchi, Inria, jusqu'au 31 mars
Brigitte Mignonnet, Interim, du 1er avril au 31 août
Nathalie Gaudechoux, Inria, depuis le 1er septembre

Personnel PRISM, Université de Versailles–Saint-Quentin

Olivier Temam, Maître de conférences

Chercheurs post-doctorant

Sylvain Lelait, ATER, Université d'Orléans, jusqu'au 31 août

Chercheurs doctorants

Laurent Challier, boursier MESR, Université d'Orléans, jusqu'au 31 octobre
Min Dai, bourse COPROR à partir du 1er octobre, Université de Versailles–Saint-Quentin
Antoine Sawaya, boursier INRIA, Université de Versailles–Saint-Quentin

Stagiaires

Bruno Blanchet, stage de DEA, ENS Ulm, du 1er mars au 31 août
Rémi Cassier, stage de fin d'études de l'ENSTA, X-ENSTA, du 1er mars au 30 juin
Jérôme Santini, stage de DEA, Université d'Orléans, du 1er avril 1996 au 16 septembre

2 Présentation de l'avant-projet

L'avant-projet A3 a démarré en 1996, à la suite de la restructuration de l'ex-action CHARME. Les recherches portent sur l'analyse de programmes, avec ses applications en optimisation de code, optimisation de la *performance* sur les nouvelles générations d'ordinateurs et optimisation du temps de développement ainsi que de la *fiabilité* des codes.

Un point crucial dans l'élaboration des processeurs performants est actuellement la recherche du meilleur compromis entre le logiciel et le matériel. Alors que la balance penche aujourd'hui en faveur de prises de décision *à l'exécution*, avec de nombreux mécanismes dynamiques, il apparaît que le prochain saut en performances ne pourra probablement se faire qu'au prix de l'abandon de certains de ces mécanismes matériels et leur renvoi au logiciel et à l'utilisateur. Dans cette optique, nous élaborons des outils destinés à être utilisés par le compilateur ou l'utilisateur pour analyser et transformer les codes pour exploiter au mieux les spécificités architecturales de la machine. Une de nos préoccupations est de coller de près aux besoins des utilisateurs, en travaillant sur des codes réels.

Au delà des performances, l'analyse statique de code est à la base des techniques classiques de génie logiciel et de réingéniérie de code. La partie plus fondamentale de nos recherches en analyse de code trouve là des applications directes.

L'avant-projet A3 a pour vocation de devenir un projet commun INRIA–Université de Versailles–Saint-Quentin, par ses fortes collaborations avec le laboratoire PRISM de cette université.

3 Actions de recherche

3.1 Analyse de code

3.1.1 Interprétation abstraite interprocédurale sur des domaines numériques relationnels

Participants : Alain Deutsch, Rémi Cassier

Il s'agit ici d'effectuer l'interprétation abstraite de procédures sur des domaines sémantiques numériques relationnels. Cette étude a initialement été motivée par une application précise : le placement automatique optimal de tests de bornes de tableaux dans des logiciels système et réseau (FOXNET) écrits dans des langages de haut niveau (Standard ML). Il s'agit là d'un problème intéressant théoriquement et pour ses applications. Parmi celles-ci, nous nous sommes pour l'instant intéressés plus particulièrement à l'élimination statique de tests dynamiques (par exemple les tests de bornes de tableaux). Nous avons implanté le prototype correspondant. Des résultats très satisfaisants ont été obtenus qui montrent la faisabilité de l'approche. Une des conclusions de l'étude est que les performances de programmes optimisés par notre méthode (et donc fiables) sont très proches de celles des programmes non fiables obtenus en supprimant la détection dynamique d'erreurs de bornes de tableaux.

3.1.2 Complexité de l'analyse d'échappement (*escape analysis*) et garbage-collection statique

Participants : Alain Deutsch, Bruno Blanchet

L'analyse d'échappement (escape analysis) est une interprétation abstraite pour programmes fonctionnels fortement typés élaborée par Park et Goldberg. Cette analyse permet de remplacer statiquement des allocations en tas par des allocations en pile, ce qui diminue le coût de la *garbage-collection* et améliore la localité des données (cache en particulier).

Park & Goldberg ont donné une borne de complexité *exponentielle* pour l'analyse d'une fonction du premier ordre, par analogie avec la *strictness analysis*. Nous montrons que cette borne supérieure est une approximation très grossière : nous exhibons en effet une borne quasi-linéaire (selon la terminologie de Gurevitch & Shelah) en $O(n \log^2 n)$. Toutefois, cette borne n'est valide qu'au premier ordre :

nous avons ensuite montré que l'*escape analysis* devient exponentielle au second ordre, et plus précisément DEXPTIME-difficile. Nous avons réalisé un premier prototype qui est intégré à une version expérimentale du compilateur CAML CSL de Xavier Leroy (projet CRISTAL). Ce travail fait l'objet d'une communication à POPL'97 [2].

Bruno Blanchet, dans le cadre de son DEA, a ensuite étendu ce prototype de façon importante en traitant les type inductifs, les constructions impératives et (avec approximations) les fonctions d'ordre supérieur, ainsi qu'en donnant une sémantique exacte non instrumentée. Les résultats obtenus par cette analyse sont tout à fait satisfaisants. L'expérimentation, sur des programmes comme le compilateur CAML CSL ou le système COQ (environ 60 000 lignes), a montré la faisabilité de l'analyse. Son coût permet de l'appliquer à tous les programmes au prix d'un surcoût à la compilation tout à fait acceptable, qui ne dépasse pas 20%. Elle a montré aussi l'intérêt de l'allocation en pile, puisqu'on obtient des gains de temps importants : souvent 5 à 20%, quelquefois plus dans des cas particulièrement favorables.

3.1.3 Analyse de flot de données pour des langages à tableaux

Participants : Laurent Challier, Alain Deutsch

Les années précédentes, nos études avaient porté sur l'analyse de dépendances de données dans les boucles, et plus précisément sur les algorithmes de programmation paramétrique en entiers, avec des résultats nouveaux sur une extension de l'algorithme de Fourier-Motzkin aux entiers. Cette analyse est en général effectuée pour des programmes à contrôle statique. Nous nous sommes orientés cette année vers une extension à des structures plus dynamiques, en nous fondant sur une sémantique opérationnelle d'un langage à tableaux. Les premiers résultats sont décrits dans [6].

3.2 Parallélisme d'instructions

Cet axe concerne le parallélisme entre instructions, dit aussi "à grain fin". Nous étudions plus particulièrement les boucles dans les programmes, et l'utilisation du *pipeline logiciel* pour l'optimisation. Les techniques décrites ici peuvent intervenir à plusieurs niveaux. Elles peuvent être intégrées dans un compilateur optimisant (paragraphe 4.2.1). Elles peuvent aussi être implantées comme outil de transformation de code pour l'utilisateur (paragraphe 3.2.4).

Comme le parallélisme d'instructions est relativement large dans les processeurs superscalaires récents, et que les problèmes d'optimisation concernent surtout les accès mémoire, nous nous intéressons plus particulièrement aux problèmes d'allocation des variables dans les registres.

3.2.1 Approche exacte pour l'optimisation des boucles avec contraintes de registres

Participants : Christine Eisenbeis, Antoine Sawaya

Cette partie concerne une approche par programmation linéaire en nombres entiers (PLNE) de l'optimisation des boucles sous contraintes de registres. Une des propriétés importantes de l'allocation des registres dans les boucles sans branchement est qu'il faut et il suffit de contrôler le nombre R de variables en vie à chaque instant pour assurer a posteriori l'allocation de ces variables sur R registres. Ceci se fait au prix d'un dépliage de la boucle (paragraphe 3.2.2).

Notre formulation prend en compte les contraintes de dépendances, les problèmes de repliage de la boucle et la contrainte du nombre de registres. Elle permet aussi prendre en compte des contraintes de ressources complexes, comme des tables de réservation quelconques [9, 4]. Cette année, nous avons mis en évidence les avantages de cette approche par rapport à une autre approche par PLNE, basée seulement sur le nombre d'itérations de la durée de vie des variables (contraintes de *buffers*) [7].

A. Sawaya a implémenté un logiciel fondé sur cette approche, qui recherche les paramètres optimaux (latence, débit) pour une configuration architecturale donnée (nombre de registres, tables de réservation). Ce logiciel sera couplé au restructureur de langage à haut niveau (paragraphe 3.2.4) et intégré à l'outil SALTO de l'IRISA (paragraphe 4.2.1).

3.2.2 Allocation de registres et déroulage de boucles

Participants : Christine Eisenbeis, Sylvain Lelait

Nous continuons à nous intéresser au problème du dépliage de boucles pour l'allocation de registres, dont nous savons qu'il peut permettre de ne pas utiliser plus de registres que le nombre maximal de variables simultanément en vie à chaque pas de temps. Comme, en contrepartie, le dépliage peut coûter cher en taille de code engendré et en nombre de défauts de cache ou de tampon d'instructions, nous étudions plus particulièrement l'influence du dépliage de la boucle sur le nombre chromatique du graphe d'interférences résultant. Après avoir défini l'an dernier une formulation unifiant l'allocation de registres et le dépliage de boucles, nous avons implanté cette année des heuristiques que nous avons pu tester [3]. Les résultats expérimentaux ont confirmé l'adéquation de notre modélisation avec le problème du dépliage minimal de la boucle pour obtenir une allocation de registres optimale.

Cependant, les temps de calcul de ces heuristiques peuvent parfois être assez importants. C'est pourquoi, grâce à l'expérience acquise avec le *meeting graph*, nous avons établi une nouvelle heuristique en collaboration avec le professeur Gao de McGill University. Cette nouvelle méthode est adaptée aux méthodes d'ordonnancement actuelles en ne prenant en compte que les morceaux de durées de vie qui durent moins d'une itération de la boucle. L'implantation de cette méthode, inspirée de celles utilisant le *meeting graph*, est en cours au sein de la plate-forme d'expérimentation MOST développée à McGill University.

3.2.3 Couplage ordonnancement et allocation de registres

Participants : Christine Eisenbeis, Sylvain Lelait, Jérôme Santini

L'idée de cette étude est de rapprocher la décomposition en circuits du meeting graph (paragraphe 3.2.2) de la décomposition en chaînes du graphe de dépendances de données dans les travaux de Berson (Pittsburgh University). Une chaîne correspond à une série d'opérations pouvant utiliser la même ressource. Le nombre minimal de chaînes est ainsi une borne supérieure du taux de parallélisme présent dans le programme. Si le nombre d'unités fonctionnelles est plus petit, l'introduction de nouvelles contraintes de succession diminue ce nombre de chaînes pour rendre le code exécutable sur l'architecture. On utilise le même principe pour les registres, en désignant au préalable pour chaque variable la dernière opération qui l'utilise (*killing instruction*). Lors de son stage de DEA, Jérôme Santini a étendu cette étude aux boucles, avec des circuits possibles dans le graphe de dépendances. Les chaînes deviennent des circuits, et l'allocation de registres se traduit par une décomposition en circuits, comme dans le *meeting graph* [5].

3.2.4 Pipeline logiciel à haut niveau

Participants : Min Dai, Christine Eisenbeis, Antoine Sawaya, Olivier Temam

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 oeuvre 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 *cache miss* lors de son chargement. C'est pourquoi nous avons intégré un restructureur de code par pipeline logiciel dans l'environnement de parallélisation Sage++ (développé à l'Université d'Indiana). Le code FORTRAN est tout d'abord découpé en code à 3 adresses, avec assignation des variables temporaires dans des tableaux. 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 simuler 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.

4 Actions nationales et internationales

4.1 Actions nationales

Nous collaborons avec l'équipe CAPS de l'IRISA, en particulier dans le projet ESPRIT OCEANS (section 4.2.1), ainsi qu'avec le laboratoire PRISM de l'Université de Versailles–Saint-Quentin.

Ch. Eisenbeis, S. Lelait et A. Sawaya participent au pôle PRS "Ordonnancement", qui regroupe aussi des équipes de l'ENST Paris, l'IMAG de Grenoble, l'INRIA de Sophia-Antipolis, l'INT d'Évry, le LITP à Jussieu et l'UTC de Compiègne. Le thème du pôle est l'étude des problèmes d'ordonnancement liés à l'exploitation et l'optimisation des machines parallèles.

Ch. Eisenbeis et F. Thomasset (avant-projet OSCAR) ont organisé les premières "*Rencontres sur les problèmes linéaires en nombres entiers*", du 28 au 30 octobre 1996, à Rocquencourt.

A. Deutsch collabore avec C. Faure (projet SAFIR, INRIA Sophia-Antipolis) sur le thème de la différentiation automatique.

4.2 Actions internationales

4.2.1 Europe de l'ouest

Cette année a vu l'acceptation du projet ESPRIT OCEANS (Optimizing Compilers for Embedded ApplicatioNS) 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). Il s'agit d'appliquer nos techniques bien aguerries de parallélisation des boucles et d'ordonnancement d'instructions aux processeurs VLIW récents, en particulier le processeur VLIW TriMedia de Philips (Pays-Bas) qui est aussi partenaire dans le consortium. Nos outils vont être intégrés dans le logiciel SALTO (System for Assembly Language Tools and Optimisation) développé dans l'équipe CAPS à l'IRISA.

Nous collaborons avec l'Université d'Edinburgh (Royaume-Uni), l'Université Polytechnique de Catalogne (Espagne) et l'Université de Versailles–Saint-Quentin autour de la création d'outils d'analyse et d'optimisation de la hiérarchie mémoire pour les utilisateurs de processeurs superscalaires.

A. Krall, de l'Université Technologique de Vienne (Autriche) a donné un séminaire "Improving Branch Prediction and Alignment by Code Replication" lors de sa visite le 20 juin 1996. De plus, S. Lelait a rejoint depuis septembre 1996 son équipe à Vienne (Autriche) et continue à collaborer avec nous.

Ch. Eisenbeis a été invitée à l'Université de Jena (Allemagne) les 1er et 2 juillet 1996 pour établir les bases d'une collaboration autour de l'analyse de code assembleur.

4.2.2 Europe de l'est

Ch. Eisenbeis participe à un projet de l'Institut Lyapunov (responsable Y. Jegou du projet CAPS à l'IRISA), commun avec l'Institute for System Programming (Moscou) et l'Institute of Applied Mathematics (Moscou) autour d'un environnement de programmation parallèle. Ce projet a reçu cette année un financement INTAS.

4.2.3 Amérique

L. Hendren, professeur à *McGill University* (Montreal) était dans le jury de thèse de S. Lelait. A l'occasion de son séjour en France, elle a donné un séminaire ("Pointer Analysis in the McCAT Compiler*") le 15 janvier 1996.

J.-L. Gaudiot, de l'*University of South California* à Los Angeles était en visite à l'INRIA et a donné un séminaire ("Structures de données pour langages fonctionnels") le 19 février 1996.

G.-R. Gao, professeur à *McGill University* (Montreal) a visité notre équipe et présenté une conférence ("Exploitation of Instruction Level Parallelism through Hardware and Software Co-Scheduling") le 22 février 1996.

S. Lelait a séjourné du 01/04 au 05/05 à McGill University, Montréal, à l'invitation du Professeur G.R. Gao.

5 Diffusion des résultats

5.1 Actions d'enseignement

A. Deutsch est professeur à l'ENSTA où il donne des cours de compilation et de programmation fonctionnelle, et membre du Conseil de Formation de l'ENSTA. Il enseigne également dans le DEA de Sémantique, Preuves et Programmation, commun a l'ENS, Paris VI, Paris VII, Orsay et au CNAM, où il assure le module d'Interprétation Abstraite.

S. Lelait était ATER à l'Université d'Orléans jusqu'au 31 août 1996. Il y a assuré des cours d'algorithmique, de C, ainsi que des travaux dirigés de Pascal et de structures de données.

A. Sawaya enseigne en $2^{ème}$ année de Sciences Économiques à l'Université de Versailles–St-Quentin-En-Yvelines (Informatique et Techniques Quantitatives pour l'économie).

5.2 Participation à des colloques

Des membres de l'équipe ont participé à des conférences et *workshops* ; on se reportera à la bibliographie pour en avoir la liste.

A. Deutsch était membre du comité de programme de la conférence SAS'96 (Third International Static Analysis Symposium) qui s'est déroulée du 24 au 26 Septembre 96 à Aix-la-Chapelle (Allemagne).

Ch. Eisenbeis était l'organisatrice locale du workshop "Instruction Level Parallelism" lors de la conférence internationale EuroPar' 96 tenue à Lyon du 26 au 28 août 1996, dont elle était également membre du comité de programme. Elle était aussi membre des comités de programme des conférences internationales PACT' 96 (International Conference on Parallel Architectures and Compilation Techniques, Boston, 20-23 octobre 1996) et MICRO' 29 (29th Annual International Symposium on Microarchitecture, Paris, France, 2-4 décembre 1996).

5.3 Conférences invitées, tutoriels, cours, etc.

Ch. Eisenbeis et A. Sawaya ont donné des présentations lors d'une journée du pôle PRS "Ordonnancement" organisée à l'INRIA de Sophia le 18 janvier 1996 (Ch. Eisenbeis : "Ordonnancement d'instructions : branchements conditionnels, registres.", A. Sawaya : "Un programme linéaire en nombres entiers pour ordonnancer les boucles sous contraintes de registres.").

Ch. Eisenbeis était invitée à donner un séminaire le 2 juillet à l'Université d'Iena (Allemagne) ("Register Allocation and Loop Scheduling").

Ch. Eisenbeis et A. Sawaya étaient invités au séminaire sur la parallélisation de boucles, organisé au château de Dagstuhl par C. Lengauer, M. Wolfe, L. Thiele et H. Zima du 15 au 19 avril 1996 (conférence de Ch. Eisenbeis : "Software Pipelining under Register Constraints").

Ch. Eisenbeis et A. Sawaya étaient invités au *Workshop on Compilers for Parallel Computers*, du 11 au 13 décembre 1996, à Aachen, Allemagne (conférence de A. Sawaya : "Variations on loop optimization and registers constraints").

Ch. Eisenbeis était membre du jury de la thèse de Sylvain Lelait soutenue le 16 janvier à l'Université d'Orleans. Elle est également rapporteur de la thèse de Damien Gautier de Lahaut (LRI, Université Paris XI) soutenue le 20 décembre.

6 Publications

Thèses

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

Communications à des congrès, colloques, etc.

[2] A. DEUTSCH, «On The Complexity of Escape Analysis», *in : 24th Annual ACM Symp. on Principles of Programming Languages*, ACM Press, Paris, janvier 1997.

[3] S. LELAIT, B. MARMOL, «Allocation cyclique de registres et déroulage de boucles», *in : Actes des Huitièmes Rencontres Francophones du Parallélisme, RenPar'8*, R. Castanet, J. Roman (réd.), p. 145–148, Bordeaux, 20-24 mai 1996.

[4] A. SAWAYA, «Code optimal sous contraintes de ressources et de registres», *in : Actes des 8èmes Rencontres francophones du Parallélisme (RenPar' 8)*, R. Castanet, J. Roman (réd.), p. 157–160, Bordeaux, 20-24 Mai 1996, ftp://ftp.inria.fr/INRIA/Projects/a3/sawaya/.

Rapports de recherche et publications internes

[5] J. SANTINI, *Ordonnancement et allocation de registres: extension de la technique URSA aux boucles*, rapport de DEA, Université d'Orléans, septembre 1996.

Divers

[6] L. CHALLIER, «Operational Semantics of a Simple Imperative Language with Arrays and Data Dependence», Document Interne, juin 1996.

[7] C. EISENBEIS, A. SAWAYA, «Optimal Loop Parallelization under Register Constraints», soumis, 1996.

[8] G. R. GAO, C. EISENBEIS, J. WANG, «Introduction to ILP Workshop», EuroPar' 96 Parallel Processing, août 1996, LNCS 1124, Springer.

[9] A. SAWAYA, «Code optimal sous contraintes de ressource et de registres», 1996, version journal de [4], soumis à TSI.

7 Abstract

The A3 group addresses program analysis, with its applications to code optimisation on instruction-level parallel processors and to software engineering. In code analysis, a new method for static elimination of dynamic tests has been designed; we have improved the complexity of the problem of escape analysis; we have developed a software based of this result, that performs efficient static garbage collection and decrease computation times of some CAML programs up to 20%. In the domain of code optimisation, emphasis has been put on the design of a high level software pipeliner, the design and experiments of

our exact method of software pipelining under register constraints, as well as the improvement of the performance of our heuristics for loop unwinding and register allocation. A new study that combines register constraints and loop scheduling has been started.