INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Projet CAPS

Compilation, architectures des processeurs superscalaires et spécialisés

Rennes

THÈME 1A

Table des matières

1 Composition de l'équipe **3**

2 Présentation et objectifs généraux **4**
2.1 Architecture de processeurs 4
2.2 Environnements de développement pour architectures hautes performances . . . 6

3 Fondements scientifiques **7**
3.1 Panorama 7
3.2 L'exécution spéculative 7
3.3 Simulation de processeurs et collecte de traces 8
3.4 Compilation pour architectures hautes performances 9

4 Domaines d'applications **13**

5 Logiciels **13**
5.1 Panorama 13
5.2 Salto : un environnement de transformations pour les langages d'assemblages (cf. 2.2) 13
5.3 Calvin2+DICE: génération de traces au vol pour la simulation de microarchitecture 14

6 Résultats nouveaux **15**
6.1 Architecture de processeurs (cf. 2.1) 15
6.2 Environnement pour architectures hautes performances (cf. 2.2) 18

7 Contrats industriels (nationaux, européens et internationaux) **23**
7.1 System level Methods and Tools Medea n° $199C9010031308011(1999-2000)$. 23
7.2 Modélisation d'architecture de microprocesseurs multithreadés pour le multimédia, CEA, (1999-2002) 23
7.3 Analyse et évaluation de performance de code pour architectures embarquées hautes performances (2000-2003) 24
7.4 Infrastructure flexible pour l'ordonnancement et l'optimisation de code (2000-2003) 24
7.5 Compilation et puissance dissipée (2000-2003) 24
7.6 Convention avec la société ACE 24

8 Actions régionales, nationales et internationales **24**
8.1 Actions régionales 24
8.2 Relations bilatérales internationales 24

9 Diffusion de résultats **24**
9.1 Animation de la communauté scientifique 24
9.2 Enseignement universitaire 25

9.3 Participation à des colloques, séminaires, invitations 25
9.4 Divers 25
9.5 Distinction 25

10 Bibliographie **26**

1 Composition de l'équipe

Responsable scientifique

André Seznec [DR Inria, en disponibilité jusqu'au 25 février 2000 chez COMPAQ]

Assistante

Huguette Béchu [TR Inria]

Personnel Inria

Pierre Michaud [CR]

Personnel UMR 6074

François Bodin [professeur, université de Rennes 1]

Jacques Lenfant [professeur, université de Rennes 1]

Ingénieurs experts Inria

Gilles Pokam [à partir du 15 avril 2000]

Chercheurs doctorants

Ronan Amicel [bourse MENESR]

Laurent Bertaux [bourse CIFRE Thomson MMD, à partir du 1er octobre 2000]

Romain Dolbeau [AMN]

Laurent Morin [bourse CIFRE STmicroelectonics, à partir du 1er octobre 2000]

Karine Heydemann [élève de l´école normale supérieure de Cachan, depuis le 1er septembre 2000]

Thierry Lafage [bourse Inria-région]

Antoine Monsifrot [bourse Inria]

Jonathan Perret [bourse Inria]

2 Présentation et objectifs généraux

Le projet Caps a pour objectif d'étudier les concepts à la fois matériels et logiciels entrant dans la conception des systèmes hautes performances.

Les performances théoriques des calculateurs croissent régulièrement. Cependant cet accroissement des performances de crête se poursuit au prix d'une complexité matérielle de plus en plus élevée. Ainsi, de nombreux niveaux de parallélisme sont présents sur le matériel, et l'obtention de performances élevées nécessite l'exploitation simultanée de tous ces niveaux par les applications. La mise au point des applications pour la performance devient de plus en plus une activité de haute technologie.

Les recherches menées au sein du projet Caps visent à exploiter de manière efficace les différents niveaux de parallélisme présents dans les applications et sur les architectures tout en masquant la complexité des matériels et systèmes à l'utilisateur.

Nos recherches en architecture de processeurs s'appuient sur une activité de veille technologique diffusée depuis 1991. Ces recherches visent à améliorer le comportement de la hiérarchie mémoire et augmenter le parallélisme d'instructions présenté au matériel. Ainsi, de nouvelles structures matérielles d'antémémoires sont étudiées afin de réduire les pénalités engendrées par les accès à la mémoire principale. D'autre part, nous étudions de nouveaux mécanismes de prédiction de branchements afin d'augmenter le parallélisme d'instructions soumis au matériel par **un** processus. Cependant, nous explorons aussi l'approche orthogonale, dite **multiflot simultané** où les instructions présentées aux unités d'exécution sont issues de **plusieurs** processus différents.

L'obtention de performances sur un processeur passe aussi par une maîtrise logicielle du parallélisme d'instructions et de la hiérarchie mémoire. C'est pourquoi, nous étudions des techniques logicielles d'optimisation de code visant à détecter et à exploiter la localité des accès à la mémoire. Des techniques de réordonnancement de code (pipeline logiciel, déroulage de boucles,...) sont aussi développées afin de soumettre un parallélisme d'instructions important au matériel. Ces techniques sont appliquées aussi bien aux processeurs généraux qu'aux processeurs enfouis (multimédia par exemple).

Afin de masquer à l'utilisateur la complexité logicielle de l'optimisation pour la performance, il convient de lui fournir des outils adaptés pour cette optimisation dans des environnements de développement. Une partie importante de notre activité est consacrée au développement de tels environnements.

2.1 Architecture de processeurs

Mots clés : microprocesseur, Risc, antémémoire, prédiction de branchement, multiflot simultané.

Résumé : *Les progrès technologiques permettent une plus grande densité d'intégration et une plus grande fréquence de fonctionnement des composants pour les processeurs. Ainsi, il est aujourd'hui possible d'intégrer sur un même composant une dizaine d'unités fonctionnelles et une grande antémémoire fonctionnant à une fréquence de l'ordre de 1 Ghz.*

Cependant ces progrès ne se traduisent pas linéairement en un gain de performances. En effet, le temps de cycle des processeurs décroît plus rapidement que les temps d'accès à la mémoire principale, ce qui rend la performance effective du processeur de plus en plus dépendante du comportement de sa hiérarchie mémoire. De même, le parallélisme d'instructions limité des programmes (dépendances de données et contrôle) réduit les gains liés à l'exécution superscalaire.

Les actions de recherche que nous menons portent sur la structure et les optimisations matérielles et logicielles des hiérarchies mémoire, en particulier antémémoires, sur les mécanismes de lancement des instructions, en particulier prédiction de branchement ainsi que sur les structures de processeur multiflot simultané. Ces actions de recherche s'appuient sur une veille technologique sur les microprocesseurs menée depuis 1991.

L'évolution des microprocesseurs est extrêmement rapide. Depuis début 1991, nous menons une activité de veille technologique et de diffusion d'informations sur les microprocesseurs. Sept rapports détaillés comparant les architectures de processeurs ont été diffusés à ce jour. Cette activité de veille technologique permet d'orienter les recherches du projet en architecture de processeurs.

La différence entre temps d'accès à l'antémémoire sur le composant et temps d'accès à la mémoire principale tend à croître. Il est donc de plus en plus important d'optimiser le comportement des antémémoires. Le taux de succès lors des accès à une antémémoire dépend de nombreux facteurs liés à son organisation matérielle et à l'application. Nos recherches portent à la fois sur l'étude de structures d'antémémoires "skewed-associative" [4] ainsi que sur les techniques logicielles de détection et d'exploitation de la localité [5] et d'optimisation du placement de données.

L'allongement des pipelines et l'exécution superscalaire font que le délai entre le chargement d'une instruction et son exécution correspond aujourd'hui à l'exécution de plusieurs dizaines d'instructions. Or, toute instruction de branchement rompt le flot de contrôle et devrait donc en principe arrêter le séquencement. Afin d'éviter un tel arrêt, des mécanismes d'anticipation appelés *prédicteurs de branchement* sont mis en œuvre dans les processeurs d'aujourd'hui. D'autre part, avec l'avènement de l'exécution dans le désordre et de l'exécution spéculative très agressive, le chargement en parallèle d'un seul bloc de base (c'est-à-dire l'anticipation d'un seul branchement par cycle) apparaît comme une limitation. Il est maintenant nécessaire de charger plusieurs blocs de base par cycle. Nos travaux dans ce domaine visent à améliorer la précision de la prédiction de branchement ainsi qu'à augmenter le nombre d'instructions chargées par cycle [11].

Si jusqu'à présent, la recherche de la performance ultime sur un seul processus a guidé l'industrie du microprocesseur, l'énorme potentiel d'intégration aujourd'hui disponible permet d'envisager que, d'ici à quelques années, plusieurs processus s'exécutent en parallèle sur le même composant. Parmi les solutions exploitant ces nouvelles données technologiques, le *multiflot simultané* [TEL95], semble l'une des méthodes les plus prometteuses. Le *multiflot simultané* est

[TEL95] D. Tullsen, S. Eggers, H. Levy, «Simultaneous multithreading : maximising on-chip parallelism», *in : 22nd Annual International Symposium on Computer Architecture*, p. 392–403, juin 1995.

basé sur l'exécution de plusieurs flots d'instructions indépendants ou issus d'une application parallèle sur un processeur superscalaire. Nous étudions les implications de l'utilisation du multiflot simultané dans le processeur.

2.2 Environnements de développement pour architectures hautes performances

Mots clés : optimisations de code, parallélisme, "tuning" d'applications, systèmes embarqués, VLIW.

Résumé : *Exploiter efficacement un système dépend fortement des environnements de programmation. Il s'agit, entre autres, de mettre en œuvre des techniques de génération et d'optimisation de code qui cachent à l'utilisateur la complexité matérielles. Les systèmes visés sont fondés sur des processeurs superscalaires ou VLIW.*

Les actions de recherches que nous menons visent à fournir aux utilisateurs des outils tels que compilateurs/optimiseurs et des outils de "tuning" interactifs pour les applications nécessitant des calculs intensifs. Dans le cadre des applications embarquées, il faut de plus que les techniques développées prennent en compte des contraintes globales telles que la taille du code, la consommation d'énergie.

Pour permettre une expérimentation en grandeur réelle nous développons des infrastructures de compilation et de simulation.

Nos études abordent le problème de la génération/optimisation de code pour systèmes haute performance, fondés sur des processeurs superscalaires ou VLIW, suivant deux approches complémentaires.

La première consiste à définir des stratégies de compilation qui combinent efficacement les méthodes de transformations de code tout en prenant en compte des contraintes globales telles la taille du code, la consommation d'énergie etc. Par exemple, nous explorons les techniques de compilation itérative qui traitent de la boucle de rétroaction dans les compilateurs et permettent ainsi de mieux combiner les optimisations intervenant à différents niveaux dans les compilateurs (code source et code machine entres autres).

La seconde problématique que nous abordons traite de la capitalisation et de la réutilisation guidée d'expertise des utilisateurs dans les environnements de "tuning" de codes. En particulier, nous étudions l'utilisation des techniques de raisonnement à partir de cas pour la sélection des techniques d'optimisation et le diagnostique des codes en regard des aspects performances.

Ces études sont accompagnées par des recherches sur les infrastructures de compilation et de simulation de jeux d'instructions. Outre l'intérêt propre de ces infrastructures, elles sont nécessaires à la validation des techniques d'optimisation et de "tuning" développées. Parmi les infrastructures déjà mises en œuvre on peut citer TSF[9], un outil de transformation de codes Fortran sur architectures hautes performances et SALTO un environnement de manipulation de langage d'assemblage [10].

3 Fondements scientifiques

3.1 Panorama

Résumé : *Les activités de recherche du projet Caps s'appuient sur des bases issues des communautés scientifiques architecture et compilation. Nous avons choisi de présenter ici brièvement quelques fondements de nos recherches : les principes et défis liés à l'exécution spéculative, le problème de la simulation de processeurs et de la collecte de traces ainsi qu'un aperçu des techniques de transformation de programmes.*

3.2 L'exécution spéculative

Mots clés : prédiction de branchement, exécution spéculative.

Résumé : *Les pipelines d'exécution des processeurs superscalaires sont de plus en plus longs. Afin de limiter les ruptures de charge dans les pipelines dues aux instructions de branchement, des mécanismes de prédiction de branchement sont mis en œuvre dans les processeurs, et les instructions prédites sont exécutées spéculativement.*

Pour atteindre un niveau de performance élevé sur les processeurs superscalaires de large degré qui apparaissent aujourd'hui, il est nécessaire de charger des instructions non-contiguës en mémoire, mais aussi de rompre les chaînes de dépendances entre instructions par la prédiction de valeurs.

Les pipelines d'exécution des processeurs sont de plus en plus longs : 12 cycles sur l'Intel PentiumPro (1995), 20 cycles sur l'Intel Pentium 4 (2000). Les processeurs sont capables d'exécuter plusieurs instructions par cycle. Le séquencement des instructions devrait être interrompu à chaque instruction de branchement en attendant le calcul effectif de la condition et/ou de la cible : hors sur beaucoup d'applications, plus d'une instruction sur 5 ou 6 est un branchement.

Sur tous les processeurs superscalaires actuels, des mécanismes de prédiction de branchement sont mis en œuvre pour continuer le séquencement *spéculatif* des instructions après un branchement sans attendre sa résolution : la cible et la direction du branchement sont prédites. En cas de mauvaise prédiction, les instructions séquencées (et parfois même déjà exécutées) doivent être annulées et le séquencement est repris sur le chemin réellement utilisé par l'application. Étant donnée la très lourde pénalité payée en cas de mauvaise prédiction de branchement, la performance effective d'un processeur dépend de la précision de la prédiction. Des schémas de prédiction de plus en plus sophistiqués sont donc mis en œuvre dans les processeurs. Parmi les informations utilisées pour prédire un branchement, on peut citer l'adresse du branchement, l'historique des derniers branchements exécutés, l'historique des derniers passages dans ce branchement [Yeh93],... Cependant les recherches continuent dans plusieurs directions, parmi

[Yeh93] T. Yeh, *Two-level adaptive branch prediction and instruction fetch mechanisms for high performance superscalar processors*, thèse de doctorat, University of Michigan, 1993.

lesquelles on peut citer, la réduction des interférences sur les tables de prédiction de branchement [7] et la prédiction des branchements indirects [CHP97].

Les processeurs actuels exécutent les instructions de manière spéculative et dans le désordre. La génération actuelle de processeurs peut exécuter jusqu'à 4, parfois 6, instructions par cycle. Il est d'ores et déjà possible d'implémenter des processeurs pouvant lancer 10 voire 16 instructions par cycle. Cependant l'obtention de telles performances ne peut pas être envisagée en utilisant les mécanismes de séquencement actuels : seules des instructions consécutives sont chargées, alors que sur beaucoup d'applications, plus d'une instruction sur 5 ou 6 est un branchement. Pour permettre de réduire ce goulôt d'étranglement, il est nécessaire de prédire plusieurs branchements par cycle [11].

Une autre difficulté surgit avec la possibilité d'exécuter un grand nombre d'instructions indépendantes en parallèle. Souvent les applications n'exhibent pas ces instructions indépendantes : or l'exécution d'un programme doit respecter les dépendances entre les instructions. La prédiction de branchement est un premier accroc à ce respect des dépendances : toute instruction postérieure à un branchement est dépendante de ce branchement ; cette dépendance est «cassée» par la prédiction, mais les instructions sont validées dans l'ordre du programme. Récemment, il a été noté que le même principe pouvait être appliqué pour aussi «casser» les dépendances de données sur les programmes : on peut ainsi prédire le résultat d'une instruction ou d'un calcul d'adresse [LS96a,SVS96].

3.3 Simulation de processeurs et collecte de traces

Mots clés : collecte de traces, simulation.

Résumé : *La validation des nouvelles idées en architecture de processeurs passe par la simulation la plus précise possible du microprocesseur et de tout son environnement. Cette simulation doit être faite cycle par cycle et doit tenir compte de l'ensemble des interactions à l'intérieur du processeur. De plus cette simulation doit être faite sur des applications si possible représentatives de la charge d'un processeur dans son environnement potentiel d'utilisation.*

Deux approches sont utilisées, la simulation dirigée par l'exécution et la simulation dirigée par les traces. Nous décrivons ici ces deux approches, leurs intérêts et limitations réciproques.

Afin de valider, au niveau performance, les architectures de processeurs, la simulation est le seul outil accepté aussi bien par l'industrie que par la communauté de recherche. Cette simulation doit être faite avant le début de la conception matérielle.

[CHP97] P. Chang, E. Hao, Y. Patt, « Target prediction for indirect jumps», *in : Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997.

[LS96a] M. Lipasti, J. Shen, « Exceeding the dataflow limit with value prediction», *in : Proceedings of the 29th International Symposium on Microarchitecture*, 1996.

[SVS96] Y. Sazeides, S. Vassiliadis, J. Smith, « The performance potential of data dependence speculation and Collapsing», *in : Proceedings of the 29th International Symposium on Microarchitecture*, 1996.

Cette simulation doit être la plus précise possible et tenir compte de l'ensemble des interactions à l'intérieur du processeur. Deux approches peuvent être utilisées : la simulation dirigée par les traces et la simulation dirigée par l'exécution.

La simulation d'architecture dirigée par les traces présente l'avantage de décorréler la simulation de l'architecture de la collecte de traces [UM97]. Ainsi on pourra simuler une architecture en lui fournissant la trace de l'exécution d'une application c'est-à-dire par exemple la liste des instructions exécutées et des adresses accédées en mémoire. Cette approche a été utilisée depuis très longtemps en architecture de processeurs. Les traces peuvent être collectées soit par matériel, soit par logiciel.

La collecte de traces d'exécution par matériel (analyseur logique) a été utilisée tant que les données et instructions circulaient sur les pattes d'entrées/sorties des processeurs. Sur les processeurs actuels, la collecte de traces ne peut plus être faite de cette manière. Ce qui explique que la collecte de traces par instrumentation logicielle soit la plus utilisée par la recherche en architecture (et aussi par l'industrie). Des outils adaptés à chaque jeu d'instructions sont aujourd'hui disponibles (Pixie, Atom, spy, EEL,...). Ces outils présentent le défaut de ne pouvoir tracer qu'une seule application et ne permettent pas en général de tracer l'activité système du processeur. Enfin le ralentissement des applications tracées est considérable (facteur 10-100) et ne permet pas d'envisager le traçage réaliste d'applications de grande taille (plusieurs centaines de milliards d'instructions). Enfin, elle est inappropriée pour la simulation réaliste de processeurs permettant l'exécution spéculative (c'est-à-dire prédisant les branchements et exécutant dans le désordre) : l'exécution spéculative requiert l'accès (en lecture) aux instructions de la fausse branche ainsi qu'aux données en mémoire de l'application tracée.

La simulation dirigée par l'exécution nécessite *l'exécution* par le simulateur de l'application tracée elle-même. Cette approche permet contrairement à la simulation dirigée par les traces de simuler l'impact des instructions exécutées spéculativement. Cependant cette approche peut s'avérer extrèmement lourde puisqu'il faut être capable de simuler non seulement le code directement écrit par le développeur, mais aussi les appels à des bibliothèques dynamiques et les appels systèmes, c'est-à-dire toutes les opérations susceptibles de modifier le contenu de la mémoire associée à l'application tracée. Cette approche a été suivie dans le simulateur SimOS. SimOS [RHWG95

3.4 Compilation pour architectures hautes performances

Mots clés : hautes performances, compilation, hiérarchie mémoire, optimisation,

[UM97] R. Uhlig, T. Mudge, « Trace-Driven memory simulation: a survey », acm *Computing Surveys*, 1997.

[RHWG95] M. Rosemblum, S. Herrod, E. Witchel, A. Gupta, « Complete computer system simulation : the SimOS approach », ieee *Parallel and Distributed Technology* $n°$ 3, 1995.]*estlesimulateurcompletd'unestationmips"bootant"lesystmeIrix.L'avantagedesimosestainsidepermettredes*
Le constat global est que la majeure partie des études pour les architectures de demain sont faites sur des traces d'applications dont on a souvent réduit le volume pour permettre des temps de simulation acceptables. Ceci peut conduire à des erreurs majeures pour le dimensionnement de structures telles que prédicteurs de branchement, antémémoire ou tlbs *(cache de traduction d'adresses). Le challenge en recherche pour la simulation réaliste d'architectures de processeurs est en fait, aujourd'hui, de parvenir à simuler le comportement des applications en vraie grandeur et dans leur environnement système.*

transformation de code.

Résumé : *L'efficacité de l'exécution d'une application tant sur une machine multiprocesseur que sur un* PC *ou une station de travail dépend très fortement de la structure des programmes. Cette structure est imposée par le programmeur mais comporte des degrés de liberté que des techniques logicielles, appelées optimisations de code, peuvent exploiter pour augmenter la performance des applications. Nous présentons un rapide aperçu des techniques de transformations de code disponibles pour implémenter un compilateur optimiseur. Ces transformations peuvent être mises en œuvre tant au niveau du code source que du code machine.*

L'efficacité des mécanismes matériels pour l'exploitation de la localité des références mémoire et du parallélisme, tant au niveau multiprocesseurs qu'instruction ("Instruction Level Parallelism"), dépend très fortement de la structure des programmes. Cette structure est imposée par le programmeur mais comporte des degrés de liberté que des techniques logicielles, appelées optimisations de code, peuvent exploiter pour augmenter la performance des applications. Ces optimisations de code sont fondées sur des transformations de programmes, qui respectent la sémantique des codes, mais réorganisent les calculs pour une meilleure exploitation d'une architecture donnée.

Les transformations de code destinées à l'amélioration de performances peuvent intervenir à plusieurs étapes dans un processus de compilation. La figure 1 montre l'organisation générale d'un compilateur. Des transformations de code peuvent être effectuées aussi bien au niveau du code source qu'au niveau du code machine.

Les optimisations effectuées au niveau du code machine sont principalement les optimisations "Peephole", qui consistent à remplacer des séquences d'instructions par des séquences plus rapides, et surtout l'application des techniques d'ordonnancement de code. Cet ordonnancement doit prendre en compte les caractéristiques fines de l'architecture telles que le nombre de registres disponibles, l'usage des ressources des processeurs, etc.

Par exemple, pour l'exploitation du parallélisme d'instructions au niveau logiciel, les méthodes les plus simples se restreignent à l'exploitation du parallélisme entre les instructions d'un même bloc de base[1]. Cependant, le nombre limité d'instructions dans un bloc de base réduit l'efficacité de ce type de techniques. En pratique, surtout dans le cas des boucles, il faut extraire le parallélisme entre des instructions de plusieurs blocs de base, par exemple en utilisant la technique du pipeline logiciel. Cette technique, fondée sur l'exploitation du parallélisme disponible entre les instructions d'itérations différentes, consiste à segmenter le code des boucles d'une manière similaire à celle utilisée par les pipelines matériels.

Au niveau du code source, les transformations de programmes utilisent toutes les informations sémantiques disponibles tant au niveau du contrôle de flot que de l'usage des variables. A ce niveau, des réorganisations majeures du code peuvent être effectuées telles que par exemple le remplacement de l'appel d'une procédure par le corps de celle-ci ("inlining"). C'est aussi sur le code source que l'on peut appliquer les techniques de parallélisation automatique et

1. Une séquence d'instructions comportant un seul point d'entrée (la première instruction) et un seul point de sortie (la dernière instruction).

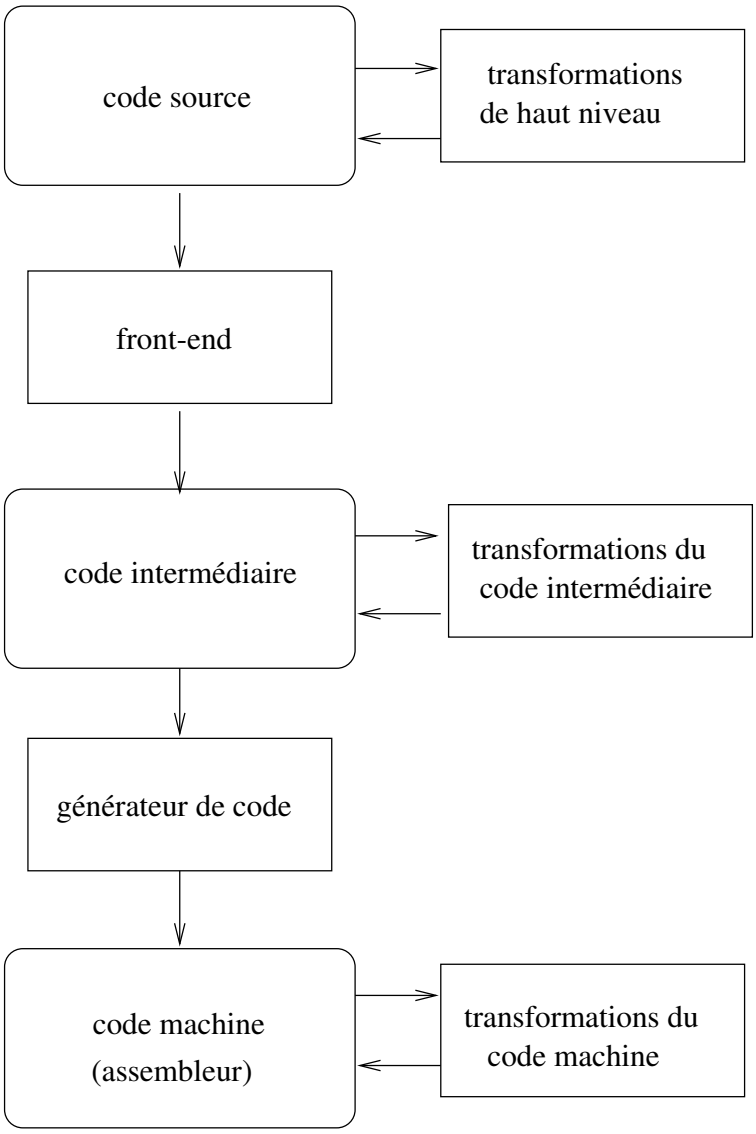


FIG. 1 – *Organisation d'un compilateur.*

les méthodes d'optimisation de la localité. Par exemple, la performance d'une hiérarchie mémoire dépend très fortement des caractéristiques de localité des accès aux données effectués par un programme. La prise en compte de la hiérarchie mémoire par un compilateur consiste à considérer les trois aspects fondamentaux suivants :

Détection et estimation de la localité : La détection de la localité est fortement liée au calcul des dépendances de données. En effet, si une dépendance existe, alors il y a réutilisation de données. Le deuxième aspect de cette question est de déterminer la proportion de références mémoire qui peuvent être évitées par l'exploitation effective de cette localité.

Exploitation de la localité : L'exploitation de la localité consiste essentiellement à déterminer le niveau de la hiérarchie mémoire qui tirera parti de la localité présente et à adapter la génération de code en conséquence.

Optimisation de la localité : Ces transformations de code ont pour but de restructurer les calculs pour permettre l'exploitation effective, par un niveau choisi de la hiérarchie, de la localité présente.

Il existe un nombre très important de transformations du code source pouvant être utilisées pour améliorer le comportement de la hiérarchie mémoire sur une application et/ou la paralléliser [BGS94]. La plupart de ces optimisations s'appliquent aux boucles. Parmi celles-ci on peut citer :

Blocage de boucles : Dans le cadre de l'optimisation de la localité, cette transformation permet de diviser l'espace d'itérations en pavés de telle sorte que les données réutilisées puissent être contenues dans un niveau de la hiérarchie mémoire.

Distribution de boucle : Les instructions d'une boucle sont réparties dans plusieurs boucles ayant le même espace d'itération que l'original. Cette transformation est utilisée pour diminuer la pression sur les registres ou extraire des calculs parallèles d'une boucle séquentielle.

Fusion de boucles : Les instructions de deux boucles sont fusionnées dans une seule boucle. Elle est par exemple utilisée pour améliorer les réutilisations de données.

Dépliage de boucle : Cette transformation consiste à répliquer le corps de la boucle. Cette transformation, toujours légale, permet de diminuer le coût de gestion de la boucle et augmente le parallélisme d'instructions potentiellement exploitable par les processeurs.

Strip-mining : Le "strip-mining" découpe l'espace d'itérations de boucle en blocs. Il permet d'ajuster la granularité des opérations dans le cas de la parallélisation ou de la vectorisation.

[BGS94] D. Bacon, S. Graham, O. Sharp, « Compiler Transformations for High-Performance Computing », *ACM Computing Surveys 26*, 4, décembre 1994, p. 345–420.

Ces transformations sont aujourd'hui relativement bien comprises individuellement. Le challenge est aujourd'hui de maîtriser l'interaction de toutes ces transformations et leur impact sur les performances. D'autres approches s'intéressent à la compilation dynamique (changement à l'exécution de binaire) ou à la compilation itérative (boucle de retour dans la compilation).

4 Domaines d'applications

Mots clés : performance, architecture de processeur, compilation, télécommunications, multimédia, biologie, santé, ingénierie, transports, environnement.

De par ses objectifs, le projet Caps travaille sur les technologies de base de l'informatique : architecture des processeurs (cf. 2.1) et compilation orientée performance (cf. 2.2). Ces travaux s'appliquent à tous les domaines d'application nécessitant de hautes performances (télécommunications, multimédia, biologie, santé, ingénierie, transports, environnement, ...). Nos travaux induisent aussi le développement de prototypes logiciels (cf. 5.2,5.3).

5 Logiciels

5.1 Panorama

Résumé : *Le projet Caps développe de nombreux prototypes logiciels de recherche : compilateurs, simulateurs, environnement de programmation,... Nous présentons ici* **Salto** *et* **Calvin2+DICE**, *2 logiciels conséquents aujourd'hui disponibles développés au sein du projet.*

5.2 Salto : un environnement de transformations pour les langages d'assemblages (cf. 2.2)

Participants : François Bodin, Laurent Bertaux, Laurent Morin, André Seznec.

Mots clés : optimisation.

Contact: François Bodin
Statut : Déposé à l'APP sous le numéro IDDN.FR.001.070004.00.R.C.1998.000.10600, disponible sur demande.

Salto propose un environnement de manipulation de programmes en langage assembleur. Une abstraction des ressources matérielles exploitables permet de les dissocier de l'algorithme d'optimisation, ce qui a deux avantages :

- le même algorithme peut être appliqué à des programmes écrits pour différentes architectures avec très peu de modifications ;
- la manipulation du code assembleur est grandement simplifiée.

Salto est composé de quatre parties :

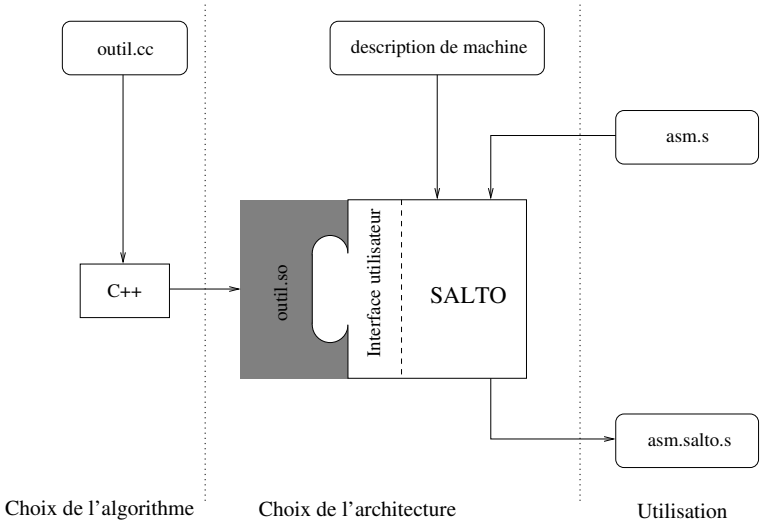
1. le noyau effectue toutes les tâches nécessaires, rébarbatives et souvent sources d'erreurs dont le programmeur a envie de se passer, notamment l'analyse lexicale et syntaxique

du code, le calcul de la structure en blocs de base et du flot de contrôle, le calcul des dépendances entre instructions ;

2. la description de la machine est un fichier qui détaille le jeu d'instructions et l'ensemble des ressources matérielles de l'architecture cible qui sont susceptibles d'intervenir dans le processus d'optimisation. Elle peut être plus ou moins précise : une description simple peut s'intéresser simplement aux unités fonctionnelles tandis qu'une description plus fine peut faire intervenir les bus d'accès à la mémoire, les ports sur le fichier de registres, etc. ;

3. l'interface utilisateur orientée objet donne un moyen d'accès aux structures de données internes de Salto. Un certain nombre de classes correspondent aux types de données connus ;

4. un algorithme d'instrumentation ou d'optimisation fourni par l'utilisateur utilise l'interface pour accéder au code et éventuellement le modifier. Salto en lui-même n'a aucun effet sur le programme assembleur, il se contente de fournir des abstractions du code et des méthodes à même de faciliter l'implantation d'algorithmes. C'est à l'utilisateur de spécialiser Salto pour obtenir un outil correspondant à ses besoins.



Pour en savoir plus, se référer à `http://www.irisa.fr/caps/projects/Salto` ou contacter François Bodin.

5.3 Calvin2+DICE: génération de traces au vol pour la simulation de microarchitecture

Participants : Thierry Lafage, André Seznec.

Mots clés : collecte de traces de programmes, simulation de micro-architecture..

Contact : André Seznec

Statut : Déposé à l'APP. sous le numéro IDDN.FR.001.470030.00.S.C.2000.000.10600 disponible sur demande.

Le système **calvin2**+DICE est composé d'une boîte à outils qui permet de générer une trace d'exécution de manière efficace et donne la possibilité d'effectuer des simulations « au vol». L'efficacité obtenue en utilisant cette boîte à outils repose sur un mode « avance rapide» de l'exécution des programmes cibles et sur la possibilité de passer dynamiquement en « mode émulé» pour effectuer des simulations.

Le mode « avance rapide » est obtenu grâce à une instrumentation légère du code cible pour uniquement exécuter le programme, sans collecter de trace ou effectuer de simulation. Ce mode doit ralentir le moins possible l'exécution car nous n'extrayons aucune information. L'outil d'instrumentation **calvin2** est utilisé pour générer le code instrumenté.

D'autre part, un émulateur de jeu d'instructions complètement intégré au programme cible (DICE) dirige un **mode émulé** qui permet de générer de la trace ou d'effectuer des simulations «au vol». L'émulateur est enfoui dans les programmes cibles de manière à avoir un accès direct à leur état et pour diriger leur exécution. Ici, l'accent est mis sur la flexibilité quant à la collecte de la trace : on peut changer de simulateur sans avoir à modifier l'émulateur ou à développer une autre infrastructure. Aussi, grâce à DICE, nous avons accès à toute l'activité utilisateur des programmes : bibliothèques partagées à chargement dynamique, code auto-modifiant, code auto-compilé.

Pendant l'exécution des programmes cibles, des changements de mode ont lieu : quand une portion de code intéressante à tracer est atteinte (en mode rapide) le mode émulé est activé et la trace est générée (ou la simulation effectuée). Le cas échéant, on passe ensuite à nouveau en mode rapide pour se positionner sur une autre portion de code intéressante à tracer.

La figure 2 illustre et récapitule le fonctionnement d'un programme cible instrumenté par **calvin2** et contenant DICE : le code instrumenté, DICE et le code de traitement de la trace font partie du même fichier exécutable et partagent le même espace d'adressage. Là, l'exécution commence en mode rapide : le code instrumenté est exécuté directement. Ensuite, un événement de basculement a lieu et l'exécution du code d'instrumentation (en pointillés) qui suit donne le contrôle à l'émulateur embarqué. Quelques instructions sont émulées, tracées et éventuellement utilisées pour une simulation, puis le contrôle retourne au mode rapide. L'apparition d'autres événements de basculement déclenche d'autres basculements en mode émulé et permet de simuler d'autres portions de code.

Sur les programmes de la suite SPEC95, le système **calvin2**+DICE introduit un facteur de ralentissement moyen de seulement 1,38 en mode rapide.

Pour plus d'informations sur le système **calvin2**+DICE, se référer à
`http://www.irisa.fr/caps/projects/calvin2DICE/index.htm` ou contacter André Seznec

6 Résultats nouveaux

6.1 Architecture de processeurs (cf. 2.1)

Participants : Thierry Lafage, André Seznec, François Bodin, Romain Dolbeau, Pierre

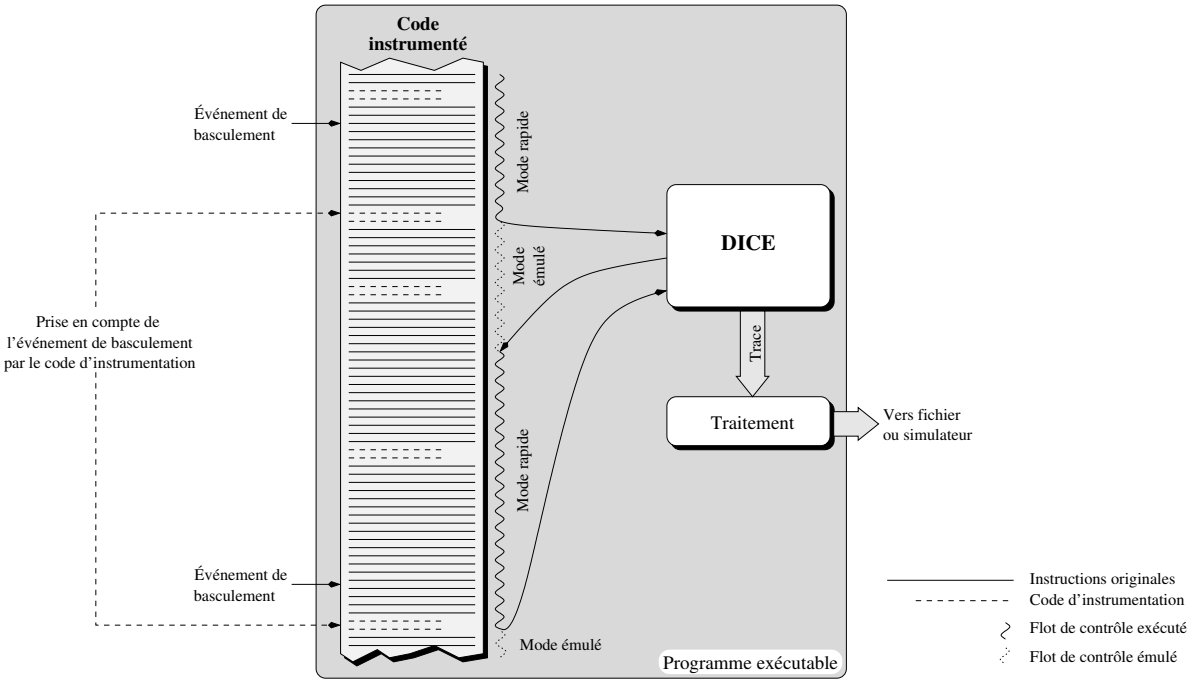


FIG. 2 – *Fonctionnement d'un programme cible traité par le système* ***calvin2****+DICE.*

Michaud.

Mots clés : microprocesseur, Risc, antémémoire, localité, hiérarchie mémoire, prédiction de branchement, multiflots.

Résumé : *Les actions de recherche du projet Caps en architecture de processeurs portent sur les mécanismes de lancement des instructions, en particulier prédiction de branchement et ordonnancement du séquencement ainsi que sur les structures de processeur multiflot simultané.*

Étude des mécanismes de séquencement

Participants : Pierre Michaud, André Seznec.

Prédicteur de branchements biaisé Les performances des microprocesseurs actuels reposent de plus en plus sur les mécanismes de prédiction de branchements dynamiques. Les tables de prédiction des branchements conditionnels sont en général mises en œuvre sans utiliser d'étiquettes, ce qui entraîne un phénomène d'interférence appelé «aliasing». Comme l'introduction d'associativité dans ces tables nécessiterait la présence d'étiquettes coûteuses, une nouvelle approche pour résoudre le problème de l'«aliasing» de conflit sans utiliser d'étiquettes a été proposée. Le Skewed Branch Predictor est une structure à trois tables : chaque

table est indexée avec une fonction différente et la prédiction est fournie par un vote à la majorité. Lorsqu'une prédiction est mauvaise, les trois bancs sont mis à jour. Cette redondance augmente l'aliasing de capacité, mais le compromis entre la réduction de l'aliasing de conflit et l'augmentation de l'aliasing de capacité s'avère bénéfique [8].

Prédiction de branchement plusieurs blocs en avance Pour permettre de réduire le goulot d'étranglement noté sur les processeurs à nombreuses unités fonctionnelles, nous avons proposé en 1996 un mécanisme appelé «multiple-block ahead branch predictor», [11]. Ce mécanisme prédit efficacement les adresses de plusieurs blocs d'instructions en avance. Un mécanisme différent appelé «trace cache» a aussi été proposé en 1996 par l'équipe de Jim Smith (université du Wisconsin) [RBS96].

Nous avons mené une étude afin de mieux comprendre les besoins en débit de chargement d'instructions dans les processeurs superscalaires à exécution non ordonnée [15]. Cette étude a montré que le besoin en débit de chargement d'instructions dépend de la fréquence des mauvaises prédictions de branchements et du parallélisme d'instructions des applications. En particulier, nous avons montré que le besoin en débit de chargement est proportionnel au degré de parallélisme d'instructions dans une fenêtre d'instructions de taille fixe, et qu'il varie également comme la racine carrée de la distance en instructions entre 2 branchements mal prédits successifs. Cette étude permet de mieux comprendre l'impact de nouvelles techniques architecturales comme la prédiction de valeur [LS96b] ou la prédication [MHB+94] sur le dimensionnement des mécanismes de chargement d'instructions. En corollaire de cette étude, nous avons constaté l'importance de disposer d'un tampon d'ordonnancement d'instructions dont la taille est du même ordre de grandeur que la distance entre les branchements mal prédits (de quelques dizaines à quelques centaines d'instructions). Cette constatation nous a conduit à orienter nos recherches vers les mécanismes d'ordonnancement dynamique des instructions.

Ordonnancement dynamique des instructions Le problème de l'ordonnancement dynamique des instructions vient de la difficulté de concilier un grand tampon d'ordonnancement avec un temps de cycle court. Nous avons commencé à explorer une nouvelle approche pour résoudre ce dilemme. Cette approche consiste à faire précéder la phase d'ordonnancement par une phase de pré-ordonnancement. Ce pré-ordonnancement est basé sur les dépendances entre instructions et les latences mais ne prend pas en compte les conflits de ressources. Cette phase de pré-ordonnancement produit une approximation de l'ordre d'exécution des instructions, ce qui facilite la tâche de la phase d'ordonnancement. Nos premiers travaux de recherche dans ce domaine nous ont conduit à proposer une mise en œuvre matérielle basée sur ce principe et à étudier sa viabilité [21].

[RBS96] E. Rotenberg, S. Bennet, J. Smith, «Trace cache : a low latency approach to high bandwidth instruction fetching», *in : Proceedings of the 29th International Symposium on Microarchitecture*, 1996.

[LS96b] M. Lipasti, J. Shen, «Exceeding the dataflow limit with value prediction», *in : Proceedings of the 29th International Symposium on Microachitecture*, 1996.

[MHB+94] S. A. Mahlke, R. E. Hank, R. A. Bringmann, J. C. Gyllenhaal, D. M. Gallagher, W. mei W. Hwu, «Characterizing the impact of predicated execution on branch prediction», *in : Proceedings of the 27th Annual International Symposium on Microarchitecture*, 1994.

Multiflot simultané et applications multimédia

Participants : Jonathan Perret, Pierre Michaud, André Seznec.

Les processeurs multiflot simultané vont devenir dans les prochaines années une brique de base des machines haute performance. Jusqu'à présent, la plupart des études sur le multiflot simultané ont porté sur des flots provenant d'applications distinctes (souvent des "petites" applications). Le comportement du mutiflot simultané a été peu étudié sur des flots provenant d'une même application.

Nous menons une étude en collaboration avec le CEA sur le comportement du multiflot simultané sur une application d'indexation et recherche d'images. Cette application grandeur nature "stresse" à la fois les capacités de calcul de la machine et sa hiérarchie mémoire. Le but de l'étude est de déterminer par des simulations comment exploiter au mieux le potentiel du multiflot simultané sur ce type d'application, en introduisant éventuellement de nouveaux supports architecturaux.

Multiflot simultané et exécution spéculative

Participants : André Seznec, Romain Dolbeau.

Parmi les solutions pour exploiter les possibilités d'intégration, le *multiflot simultané* [TEL95] est l'une des solutions les plus prometteuses. Disposer de plusieurs flots exécutables simultanément sur une architecture *superscalaire* doit permettre de maximiser le taux d'utilisation du processeur et donc les performances. Cette approche a d'ailleurs été retenue pour un futur processeur haute perormance de Compaq, l'Alpha 21464. Une étude a été menée pour quantifier le comportement des architectures *multiflots simultanés* en présence de changements de contexte [18].

Mais le multiflot seul n'apporte aucun bénéfice si la charge de travail du processeur se limite à une application non parallélisée. Une nouvelle voie étudiée est la création de *flots spéculatifs* appartenant à l'application séquentielle. Ces "processus spéculatifs" sont prédits lors de l'exécution (par exemple, les itérations suivantes d'une boucle, le retour d'une procédure lancée *avant* l'exécution de cette procédure, ...) et commencent leur exécution parallèlement (simultanément) au flot non spéculatif. Les travaux en cours consistent à developper les outils permettant d'étudier le potentiel de gain d'une telle approche : instrumentation des programmes et outils d'étude du parallèlisme potentiel et des dépendances pour les divers flots spéculatifs envisagés.

6.2 Environnement pour architectures hautes performances (cf. 2.2)

Mots clés : compilation, programmation parallèle, parallélisation automatique, portage d'applications, optimisation, simulation, multimédia.

Participants : Ronan Amicel, François Bodin, Laurent Bertaux, Laurent Morin, Karine

[TEL95] D. Tullsen, S. Eggers, H. Levy, « Simultaneous multithreading : maximising on-chip parallelism », *in : 22nd Annual International Symposium on Computer Architecture*, p. 392–403, juin 1995.

Heydemann, Thierry Lafage, Antoine Monsifrot, Gilles Pokam, André Seznec.

Résumé : *L'obtention de performances sur les architectures hautes performances nécessitent des outils logiciels adaptés qui cachent à l'utilisateur la complexité des matériels et des systèmes.*

Les actions de recherche que nous menons visent à fournir aux utilisateurs de calculateurs hautes performances des outils tels que compilateur, aide au portage, optimiseur pour permettre des développements et/ou portages d'applications hautes performances.

Ainsi, nous développons TSF*, un outil d'aide au portage de codes Fortran sur architectures hautes performances et Salto un environnement de manipulation de langage d'assemblage.*

Aide au portage sur les architectures hautes performances

Participants : François Bodin, Antoine Monsifrot.

L'ensemble des techniques mises en œuvre dans les compilateurs donnent généralement de bons résultats mais échouent fréquemment. De plus, toutes les techniques non conservatrices de la sémantique des codes sont interdites au compilateur. En conséquence l'activité de tuning de code reste en partie manuelle tout en étant techniquement difficile et en faisant appel à beaucoup de savoir faire.

L'approche développée vise à accélérer cette activité grâce à l'utilisation conjointe de techniques issues de deux domaines : l'analyse statique et dynamique des programmes et le raisonnement à partir de cas («Case-Based Reasoning»).

Un prototype est en cours de développement au dessus de TSF. TSF [9] est une infrastructure de transformation de programmes qui est construite au dessus de FORESYS, un outil de la société Simulog qui fournit à TSF l'infrastructure de base à l'analyse de programmes Fortran. Le prototype a pour objectif d'aider au choix des transformations adaptées en s'appuyant sur des expériences d'optimisation de situations similaires répertoriées.

TSF permet de calculer une représentation abstraite des nids de boucles qui nous permet de déduire pour chacun d'eux un ensemble d'indices. Ces indices informent le système sur les accès de tableaux et les dépendances de données afin de connaître les propriétés de la boucle par rapport à l'architecture du processeur. De plus, cette représentation abstraite est utilisée pour reconnaître certains motifs correspondant par exemple aux produits de matrices ou aux réductions creuses. A l'aide de ces indices, le système propose ensuite à l'utilisateur les transformations à effectuer sur le code. En procédant ainsi nous pouvons proposer des transformations de code à l'utilisateur et ainsi accélérer le tuning. TSF nous permet aussi d'instrumenter les codes pour les analyser dynamiquement. Des expérimentations sur des codes de grandes tailles sont en cours.

Compilation itérative

Participants : François Bodin, Karine Heydemann.

Les applications embarquées hautes performances posent de nouveaux défis pour la production de code optimisé. Un compilateur optimisant joue un rôle clé dans la chaîne de développement. Les optimisations permettant d'exploiter le parallélisme d'instruction, intégré dans les processeurs enfouis (VLIW), provoquent en général un accroissement significatif de la taille du code. En effet, l'amélioration d'une propriété (temps d'exécution, exploitation du parallélisme d'instruction) grâce à une optimisation s'accompagne souvent de la dégradation d'une autre (taille du code, pression sur les registres) et ainsi des compromis sont nécessaires.
Aborder cette problématique remet en cause la structure même des compilateurs classiques, qui ne permettent pas un contrôle fin des interactions entre les optimisations au niveau du code source et au niveau du code machine. En effet les optimisations, offertes par des compilateurs standards, sont appliquées localement, alors que les contraintes pour les applications enfouies sont globales.
L'approche itérative de la compilation a été validée pour l'exploration de l'espace des paramètres des optimisations et l'évaluation de l'impact/l'interaction des transformations appliquées [16]. Cependant seules des stratégies simples et deux optimisations ont été étudiées. Les travaux en cours visent à définir de nouveaux schémas de compilation permettant de respecter/optimiser des contraintes globales étendues (consommation d'énergie, comportement de l'application vis-à-vis du cache ou la taille du code). L'étude des interactions entre les transformations avec d'autres optimisations et la définition de stratégies d'exploration de l'espace des paramètres sont au coeur de ces travaux.

Utilisation des instructions multimédia

Participants : François Bodin, Gilles Pokam.

La majorité des processeurs sont aujourd'hui équipés de supports architecturaux permettant le traitement efficace d'applications multimédia. Pour ce faire, ces processeurs proposent de nouvelles instructions dédiées, appelées instructions multimédia. Leur exploitation au niveau du code source n'est cependant pas facile. En effet, les instructions multimédia ne sont souvent disponibles que sous forme de fonctions intrinsèques ou de macros prédéfinies, écrits en langage assembleur. Leur exploitation dans le code source n'est donc pas automatique, puisque cette tâche requiert une intervention manuelle qui peut s'avérer fastidieuse. De plus, l'aspect variable de ces instructions d'une plateforme à l'autre pose également des problèmes de portage de code.

Des travaux de recherche ont donc été initiés pour pallier cette carence. Leur aspect porte sur la spécification et le développement d'un module reciblable de pré-traitement de code source C. Ce pré-processeur recherche les séquences d'instructions - ou les expressions - susceptibles d'être remplacées par des instructions multimédia vectorisées, disponibles dans le langage sous forme d'instructions spécialisées (fonctions intrinsèques ou prédéfinies). Un prototype est en cours de réalisation. Ce travail s'effectue actuellement dans le cadre du contrat SMT Medea.

Analyse et évaluation de performance de code pour architectures embarquées hautes performances

Participants : François Bodin, Laurent Morin.

Un environnement de développement destiné à la mise au point de programmes multimédias embarqués nécessite une prise en compte à la fois du programme à implémenter et du code qui est exécuté. Pour cela elle doit incorporer les deux représentations de l'application – le code assembleur et le code source – et améliorer les analyses en calculant leurs relations et leurs caractéristiques. L'efficacité de l'évaluation est conditionnée par la qualité et le nombre de ces données. Ces dernières devront être complétées par des informations obtenues par simulation ou exécution du programme. Les analyses pourront aussi être perfectionnées par des modélisations basées sur plusieurs niveaux d'abstractions. Une étude en partenariat avec Thomson Multimédia vise à expérimenter une plate-forme pour des média processeurs. Les analyses doivent pouvoir prendre en compte une description complète de l'architecture cible qui paramètre l'environnement. La flexibilité pourra être augmentée par la possibilité d'intégrer des traitements ou des bibliothèques d'optimisations à l'interface de développement.

Infrastructure flexible pour l'ordonnancement et l'optimisation de code

Participants : François Bodin, Laurent Bertaux.

La production de code hautement optimisé pour des processeurs spécialisés dans le cadre d'applications embarquées hautes performances nécessite de nouveaux outils de compilation. D'un côté, il s'agit de définir des outils flexibles compatibles avec le temps de développement très court de ce type de système. De l'autre, il faut mettre en œuvre des techniques d'optimisation très sophistiquées prenant en compte les caractéristiques fines des processeurs embarqués hautes performances. Ce code très optimisé doit respecter des contraintes de taille de code, de performances, de consommation électrique et finalement les contraintes temps réel. Il s'agit de définir une infrastucture flexible pour la mise en œuvre des techniques d'ordonnancement et d'optimisation du code machine. Ce travail s'appuie sur les travaux antérieurs autour du système Salto [10].

Un des concepts de base de cette infrastructure est l'abstraction spécialisée pour permettre le développement de phases d'optimisation de manière indépendante la définition de l'architecture cible. Cette approche a pour but de simplifier la réutilisation et la mise en œuvre des algorithmes. Afin de valider cette approche, nous avons spécifié une interface de communication entre le compilateur et un allocateur de registres.

Ces travaux font l'objet d'une collaboration avec ST Microelectronics (Central R&D - site de Crolles).

Absciss : génération de simulateurs hautes performances de jeu d'instructions

Participants : François Bodin, Ronan Amicel.

La simulation de jeu d'instructions consiste à exécuter sur une machine *hôte* un programme compilé pour une machine *cible*. Le projet Absciss vise à générer automatiquement des simula-

teurs de jeu d'instructions rapides à partir d'une description de l'architecture cible. Le système est basé sur l'infrastructure SALTO, ce qui le rend reciblable. L'utilisation de la technique de *simulation compilée* permet d'atteindre des performances élevées par rapport aux méthodes interprétées traditionnelles, et donc de simuler des programmes plus gros et plus réalistes. Les applications d'un tel système sont de pouvoir évaluer différents jeux d'instructions, valider le *back-end* d'un compilateur ou tester des programmes, sans disposer d'une implémentation matérielle du processeur. Un prototype est en cours de réalisation pour l'architecture TriMedia. Les premiers tests montrent des performances nettement supérieures à celles d'un simulateur classique, ce qui valide l'approche retenue.

Collecte de traces d'exécution et sélection de tranches d'exécution représentatives

Participants : François Bodin, Thierry Lafage, André Seznec.

Dans le cadre de la simulation dirigée par la trace, les outils logiciels d'instrumentation actuellement disponibles (Pixie, Atom, ...) ont le défaut de ralentir de manière très significative l'exécution des applications (facteur 10-50), même si la trace générée n'est pas utilisée dans sa totalité. En effet, la trace d'une application est très volumineuse (plusieurs giga-octets pour de petites applications) et donc très difficile à utiliser en ce sens qu'une simulation sur une trace entière est très (trop) coûteuse en temps. Alors, des techniques telles que l'*échantillonnage de trace* sont utilisées pour réduire ce volume. D'autre part, tout programme comporte une phase d'initialisation qui n'est pas représentative de l'exécution totale. De ce fait, les premiers milliards d'éléments de trace qui correspondent à cette phase d'initialisation sont, dans la plupart des cas, ignorés.

Le ralentissement induit par l'instrumentation est trop important et pour cela ne permet pas, actuellement, de collecter des échantillons de traces sur de réelles (grosses) applications exécutées dans leur totalité. Par réelles applications, nous entendons par exemple les applications de calcul intensif, les applications de type client-serveur (X, gestion de bases de données), mais aussi le système d'exploitation lui-même qui génère une activité rarement prise en compte.

Afin d'adresser ce problème, nous avons défini une nouvelle méthode de collecte de traces qui permet de ne pas trop ralentir les parties du programme testé dont la trace sera rejetée. Pour cela, l'exécution directe d'un code cible légèrement instrumenté produit un mode d'exécution dit *rapide*. D'autre part, un émulateur de jeu d'instructions embarqué gère un mode dit *émulé* et permet la simulation. L'émulation de jeu d'instructions permet d'avoir accès à la trace de tout le code utilisateur (par opposition à celui du système d'exploitation) : bibliothèques partagées, code auto-compilé, code auto-modifiant. Aussi, en mode « émulation », la trace est aisément générée par des appels de fonctions d'analyse écrites en langage de haut niveau (C) et regroupées dans une bibliothèque partagée à chargement dynamique. Ainsi, la génération d'une trace différente nécessite uniquement de modifier les fonctions d'analyse et de régénérer la bibliothèque dynamique : les programmes cibles instrumentés et contenant l'émulateur peuvent être réutilisés tels quel.

À l'exécution, cette approche permet des changements de mode dynamiques entre le mode rapide et le mode émulé afin de ne simuler que quelques tranches d'exécution déterminées.

calvin2 et l'émulateur DICE constituent une implémentation de cette approche sur plates-formes d'architecture SPARC. Le système **calvin2**+DICE introduit un facteur de ralentissement moyen de seulement 1,38 en mode rapide sur les *benchmarks* SPEC95.

Ce faible taux de ralentissement pour l'« avance rapide » de l'exécution permet d'atteindre rapidement n'importe quelle tranche d'exécution pour la simuler. Ceci conduit naturellement au problème de la sélection de tranches d'exécution représentatives de l'exécution des programmes cibles.

Pour cela, nous caractérisons le comportement dynamique des programmes cibles pour, ensuite, extraire quelques tranches d'exécution représentatives. Grâce à la classification automatique, des ensembles de tranches d'exécution à comportements proches sont sélectionnés puis dans chaque ensemble (classe) un représentant est choisi. Des tests sur les *benchmarks* SPEC95 ont été réalisés en utilisant la méthode CHAVL de classification hiérarchique développée par I. Lerman du projet AIDA [Ler91]. Ils ont montré que, pour des simulations de caches et de prédicteurs de branchements, en simulant de 2 % à 5 % de l'activité, des résultats comparables à ceux par échantillonnage à 10% sont obtenus.

7 Contrats industriels (nationaux, européens et internationaux)

7.1 System level Methods and Tools Medea n° $199C9010031308011(1999-2000)$

Participants : François Bodin, Pierre Michaud, Gilles Pokam, André Seznec.

Dans ce projet, en collaboration avec le projet A3 de l'Inria Rocquencourt, nous concevons un module reciblable de recherche des séquences d'instructions - ou d'expressions - susceptibles d'être remplacées par des instructions multimédia vectorisées. C'est actuellement un point bloquant pour une exploitation efficace des nouvelles générations de DSP hautes performances.

7.2 Modélisation d'architecture de microprocesseurs multithreadés pour le multimédia, CEA, (1999-2002)

Participants : Pierre Michaud, Jonathan Perret, André Seznec.

La convention vise à explorer les concepts architecturaux des microprocesseurs multithreadés pour le domaine des applications multimédia, ainsi que les mécanismes de compilation et d'optimisation de code qui peuvent leur être destinés. La première étape consiste en l'analyse, la conception et la réalisation d'un simulateur d'architectures multithreadées adaptable en fonction des évolutions de la conception architecturale. La seconde étape consiste en l'analyse des variantes architecturales et de leur contrepoint logiciel, dans un souci d'optimisation de la performance orientée sur le domaine des applications multimédia (incluant les aspects traitement des images, traitement du son, bases de données, distribution vidéo, etc.).

[Ler91] I. C. Lerman, « Foundations of the Likelihood Linkage Analysis (LLA) Classification Method », *Applied Stochastic Models and Data Analysis 7*, 1991, p. 63–76.

7.3 Analyse et évaluation de performance de code pour architectures embarquées hautes performances (2000-2003)

Participants : François Bodin, Laurent Morin.

La thèse de Laurent Morin est financée dans le cadre d'une convention CIFRE avec la société Thomson MMD.

7.4 Infrastructure flexible pour l'ordonnancement et l'optimisation de code (2000-2003)

Participants : François Bodin, Laurent Bertaux.

La thèse de Laurent Bertaux est financée dans le cadre d'une convention CIFRE avec la société STmicroelectronics.

7.5 Compilation et puissance dissipée (2000-2003)

Participants : François Bodin, Gilles Pokam.

Cette étude, commençant en janvier 2001, fait l'objet d'une convention CIFRE avec la société STmicroelectronics pour le financement de la thèse de Gilles Pokam.

7.6 Convention avec la société ACE

Participants : François Bodin, Gilles Pokam.

La société ACE a mis à disposition gratuite l'usage de la chaîne de compilation COSY.

8 Actions régionales, nationales et internationales

8.1 Actions régionales

La thèse de Thierry Lafage est partiellement financées par la région Bretagne.

8.2 Relations bilatérales internationales

Dans le cadre de travaux sur l'étude de transformation de code et puissance dissipée, une collaboration est mise en place avec Université de Stanford (Pr Michelli) et l'Université de Milan (Pr Benini) et STmicroelectronics.

9 Diffusion de résultats

9.1 Animation de la communauté scientifique

– A. Seznec a été membre des comités de programme des conférences 7th International Symposium on High Performance Computer Architecture (HPCA'7), 33rd International

Symposium on Microarchitecture (Micro'33), 1st Memory Wall workshop, 4th Multithreaded Architecture workshop (MTEAC'4), du congrès africain de recherche en informatique (CARI 2000) et du 5ième symposium sur les architectures de machine (SYMPA 6).

- F. Bodin a été membre du comité de programme du First Workshop on Advanced Software for Pervasive Environments and Information and Server Appliances (ASPEISA '00) (dec. 29, 2000), National Tsing-Hua University Hsinchu, Taiwan.

- F. Bodin est membre du comité de rédaction de la revue TSI.

9.2 Enseignement universitaire

F. Bodin et P. Michaud interviennent dans les cours d'architectures et compilation du DEA informatique et du DIIC de l'université de Rennes I.

A. Seznec intervient dans un cours sur les architecture de processeurs à l'ENST de Bretagne.

9.3 Participation à des colloques, séminaires, invitations

Outre les conférences et workshops donnant lieu à publication des actes listés dans la bibliographie, les membres du projet Caps ont présenté leurs travaux dans les séminaires ou workshops suivants :

- A. Seznec a présenté une conférence sur les "architectures de processeurs: panorama et nouveaux challenges" au séminaire A3 (mai 2000), en conférence invitée au 6ème Symposium en Architectures Nouvelles de Machines à Besançon (juin 2000), au forum conjoint ORAP/SPEED-UP à Genève (octobre 2000) et au congrès africain de recherches en informatique, CARI 2000 à Antananarivo (octobre 2000).

- F. Bodin a présenté une communication invitée à la conférence AD2000: "Performance Issues in Automatic Differentiation on Superscalar Processors"

- F. Bodin a présenté ses travaux à Philips Research, Eindhoven le 25/09/2000: "Infrastructures for assembly level tools"

9.4 Divers

- A. Seznec est membre de la commission d'évaluation de l'INRIA

- F. Bodin est vice-président du comité des projets de l'IRISA.

- F. Bodin est responsable du DEA d'informatique de l'université de Rennes 1.

9.5 Distinction

Jacques Lenfant a été fait Chevalier de la Légion d'Honneur, le 30 novembre 2000.

10 Bibliographie

Ouvrages et articles de référence de l'équipe

[1] F. Bodin, P. Beckman, D. Gannon, J. Srinivas, « Sage++ : a class library for building Fortran and c++ restructuring tools », *Proceedings of the Second Object-Oriented Numerics Conference*, avril 1994.

[2] F. Bodin, W. Jalby, C. Eisenbeis, D. Windheiser, « Window-based register allocation », *Code Generation - Concepts, Tools, Techniques, Proceedings of the International Workshop on Code Generation*, 1991, p. 119–145.

[3] F. Bodin, L. Kervella, T. Priol, « Fortran-S : a fortran interface for shared virtual memory architectures», *in : Proceedings of Supercomputing*, ieee Computer Society Press (éditeur), p. 274–283, novembre 1993.

[4] F. Bodin, A. Seznec, « Skewed associativity improves performance and en hances predictability», ieee *Transactions on Computers*, mai 1997.

[5] C. Eisenbeis, W. Jalby, D. Windheiser, F. Bodin, « A strategy for array management in local memory», *Journal of Mathematical Programming*, 63, 1994, p. 331–370.

[6] S. Hily, A. Seznec, «Standard memory hierarchy does not fit simultaneous multithreading», *in : Proceedings of the Workshop on Multithreaded Execution, Architecture and Compilation (MTEAC' 98)*, Las Vegas, février 1998.

[7] P. Michaud, A. Seznec, R. Uhlig, « Trading conflict and capacity aliasing in condit ional branch predictors», *in : Proceedings of the 24th International Symposium on Computer Architecture*, ieee-acm (éditeur), Denver, juin 1997.

[8] P. Michaud, *Chargement des instructions sur les processeurs superscalaires*, Thèse de doctorat, université de Rennes I, novembre 1998.

[9] Y. Mével, *Environnement pour le portage de codes orienté performance sur machines parallèles et monoprocesseurs*, Thèse de doctorat, université de Rennes I, mars 1999.

[10] E. Rohou, *Infrastructures et stratégies de compilation pour parallélisme à grain fin*, Thèse de doctorat, université de Rennes I, novembre 1998.

[11] A. Seznec, S. Jourdan, P. Sainrat, P. Michaud, «Multiple-block ahead branch predictors», *in : Proceedings of the 7th conference on Architectural Support for Programming Languages and Operating Systems*, octobre 1996.

Thèses et habilitations à diriger des recherches

[12] T. Lafage, *Étude, réalisation et application d'une plate-forme de collecte de traces d'exécution de programmes*, Thèse de doctorat, université de Rennes I, décembre 2000.

Articles et chapitres de livre

[13] B.Chapman, J.Merlin, D.Pritchard, F.Bodin, Y.Mevel, T.Sorevik, L.Hill, «Program Development Tools for Clusters of Shared Memory Multiprocessors», *Journal of Supercomputing*, à paraître.

[14] T. Lafage, A. Seznec, «Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream», *Workload Characterization*, à paraître.

[15] P. Michaud, A. Seznec, S. Jourdan, « An exploration of instruction fetch requirement in out-of-order superscalar processors», *International Journal of Parallel Programming*, à paraître.

[16] E. Rohou, F. Bodin, C. Eisenbeis, A. Seznec, «Handling Global Constraints in Compiler Strategy», *International Journal of Parallel Programming*, août 2000.

Communications à des congrès, colloques, etc.

[17] F. Bodin, A. Monsifrot, «Performance Issues in Automatic Differentiation on Superscalar Processors», *in : Proceedings of AD2000 (Automatic Differentiation: From Simulation to Optimization)*, juin 2000.

[18] R. Dolbeau, «Multiflot simultané, changements de contexte, et effets sur les caches», *in : Actes Du Sixième Symposium Architectures nouvelles de machines*, juin 2000.

[19] T. Lafage, A. Seznec, «Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream», *in : Workshop on Workload Characterization (WWC 2000)*, septembre 2000.

[20] T. Lafage, A. Seznec, «Combining Light Static Code Annotation and Instruction-Set Emulation for Flexible and Efficient On-the-fly Simulation», *in : Euro-Par 2000*, Munich, août 2000.

[21] P. Michaud, A. Seznec, « Data-flow prescheduling for large instruction windows in out-of-order processors», *in : 7th International Conference on High Performance Computer Architecture*, janvier 2001.