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

Responsable scientifique
André Seznec [DR INRIA]

Assistante
Huguette Béchu [TR INRIA, jusqu'au 4 novembre 2002]
Evelyne Livache [TR INRIA, à partir du 5 novembre 2002]

Personnel Inria
Pierre Michaud [CR, à partir du 1er août 2002]

Personnel UMR 6074
François Bodin [professeur, université de Rennes 1]
Jacques Lenfant [professeur, université de Rennes 1]
Henri-Pierre Charles [Maitre de conférences, université de Versailles-Saint-Quentin, en délégation INRIA à partir du 15 septembre 2002]

Ingénieurs experts Inria
Stéphane Bihan
Eric Courtois
Julien Simonnet

Ingénieur associé Inria
Pierre Villalon

Chercheurs doctorants
Ronan Amicel [bourse MENRT, jusqu'au 30 septembre 2002]
Laurent Bertaux [bourse CIFRE STMicroelectronics]
Amaury Darsch [bourse INRIA, à partir du 15 janvier 2002]
Assia Djabelkhir [bourse accord franco-algérien]
Romain Dolbeau [Allocation couplée]
Antony Fraboulet [bourse INRIA]
Karine Heydemann [Allocation couplée]
Antoine Monsifrot [bourse Inria, jusqu'au 30 septembre 2002]
Laurent Morin [bourse CIFRE Thomson MMD]
Gilles Pokam [bourse CIFRE STMicroelectronics]
Olivier Rochecouste [bourse INRIA]
Pascal Terjan [bourse CIFRE STMicroelectronics, à partir du 1er octobre 2002]
Eric Toullec [bourse MENRT]

2. Présentation et objectifs généraux

2.1. Introduction

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 du matériel à l'utilisateur.

Nos recherches en architecture de processeur 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 branchement 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, etc) 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.

2.2. Architecture de processeurs

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

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 2 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. De plus, nos actions de recherche visent aussi à simplifier l'implémentation des fichiers de registres, et l'ordonnancement dynamique des instructions.

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 » [11] ainsi que sur les techniques logicielles de détection et d'exploitation de la localité [12] 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 et à augmenter le nombre d'instructions chargées par cycle [14].

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* [32], semble l'une des méthodes les plus prometteuses. Le *multiflot simultané* est 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.

La complexité des processeurs hautes performances réside aussi dans la profondeur du pipeline d'exécution (une instruction est exécutée au plus tôt à l'étage 18 sur le processeur Intel Pentium 4 !), dans les délais de communication sur un même composant (plusieurs cycles sont nécessaires pour communiquer une donnée d'une unité fonctionnelle à une autre) et dans une consommation électrique de plus en plus élevée. Nos recherches visent à simplifier la mise en œuvre du processeur tout en permettant des performances élevées.

2.3. Environnements de développement pour architectures hautes performances

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

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

Les actions de recherche 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 transformation de code tout en prenant en compte des contraintes globales telles que la taille du code, la consommation d'énergie, etc. Par exemple, nous explorons les techniques de compilation itérative qui permettent de mieux combiner les optimisations intervenant à différents niveaux dans les compilateurs (code source et code machine entre 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 diagnostic 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[4], un outil de transformation de codes Fortran sur architectures hautes performances et SALTO, un environnement de manipulation de langages d'assemblage [6].

3. Fondements scientifiques

3.1. Panorama

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*.

Les pipelines d'exécution des processeurs superscalaires sont de plus en plus longs. Afin de limiter les cycles perdus dus 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. Or sur beaucoup d'applications, 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 par ce branchement [34], ... Cependant les recherches continuent dans plusieurs directions, parmi lesquelles on peut citer la réduction des interférences sur les tables de prédiction de branchement [13] et la prédiction des branchements indirects [28].

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 de telles performances ne peuvent pas être obtenues 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 goulot d'étranglement, il est nécessaire de prédire plusieurs branchements par cycle [14].

Une nouvelle difficulté surgit avec la possibilité d'exécuter un grand nombre d'instructions indépendantes en parallèle. Souvent les applications n'exhibent pas directement 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 [29][31].

3.3. Simulation de processeurs et collecte de traces

Mots clés : *collecte de traces*, *simulation*.

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

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 [33]. 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 broches d'entrées/sorties des processeurs. Sur les processeurs actuels, la collecte de traces ne peut plus être faite de cette manière. Ceci 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. De plus 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 extrêmement 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 [30] est le simulateur complet d'une station MIPS « bootant » le système Irix. L'avantage de SimOS est ainsi de permettre de simuler un processeur avec l'ensemble de son système d'exploitation. Par contre, les performances de la simulation restent très limitées et ne permettent pas d'envisager la simulation des « grosses » applications (plusieurs centaines de milliards d'instructions).

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, mémoires cache ou TLB (cache de traduction d'adresses). Le défi 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.

3.4. Compilation pour architectures hautes performances

Mots clés : *haute performance*, *compilation*, *hiérarchie mémoire*, *optimisation*, *transformation de code*.

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 transformation 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 des processus que des instructions (« 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.

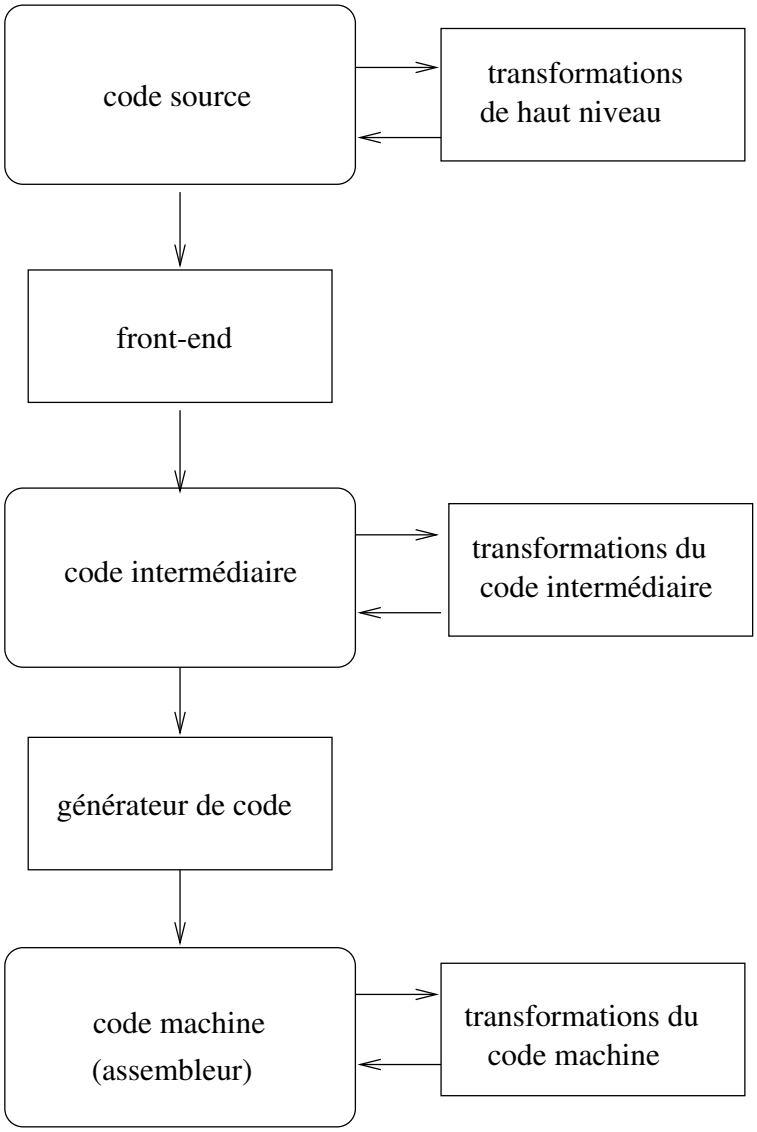

Figure 1. Organisation d'un compilateur.

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. À 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 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 d'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 [27]. 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.

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

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.

Ces transformations sont aujourd'hui relativement bien comprises individuellement. Le défi 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 (modification du binaire à l'exécution) ou à la compilation itérative (boucle de retour dans la compilation).

4. Domaines d'application

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. 2.2) et compilation orientée performance (cf. 2.3). Ces travaux s'appliquent à tous les domaines d'application nécessitant de hautes performances (télécommunications, multimédia, biologie, santé, ingénierie, transports, environnement, etc). Nos travaux induisent aussi le développement de prototypes logiciels (cf. 5.2, 5.3).

5. Logiciels

5.1. Panorama

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

5.2. Salto : un environnement de transformations pour les langages d'assemblage

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 assembleur, 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 ;
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.

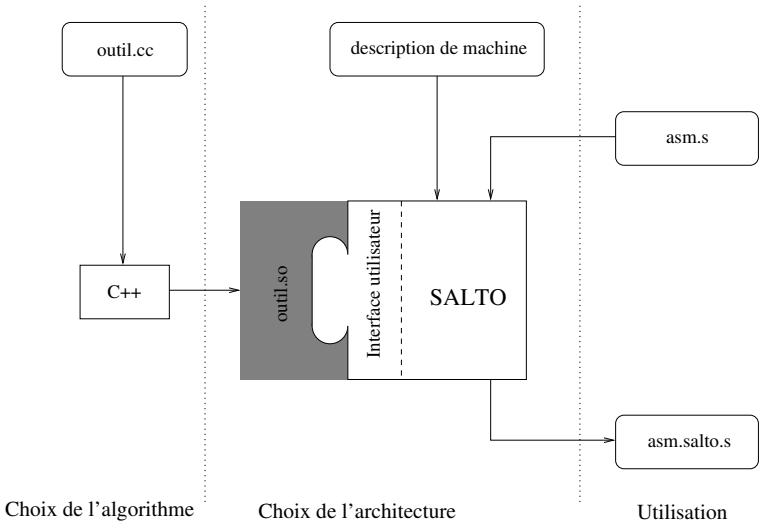

Figure 2.

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 : Pierre Villalon, 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** a été développé par Thierry Lafage au cours de sa thèse [2]. Il 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 exécuter uniquement le programme, sans collecter de trace ou effectuer de simulation. Comme nous n'extrayons aucune information, ce mode doit ralentir le moins possible l'exécution. 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.

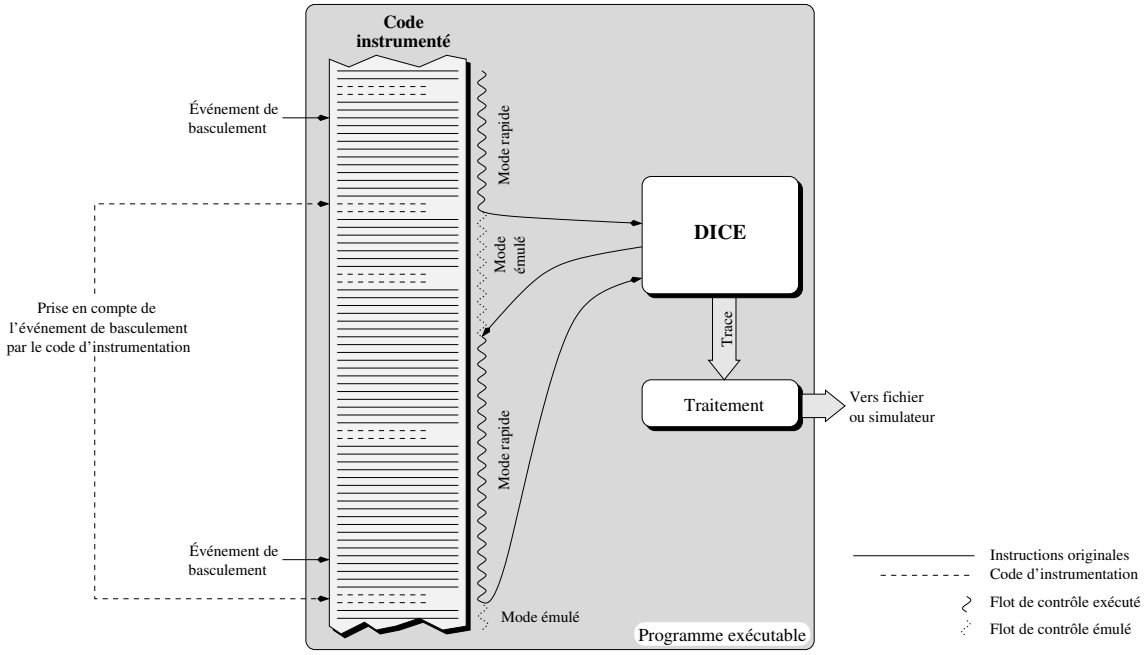

*Figure 3. Fonctionnement d'un programme cible traité par le système **calvin2**+DICE.*

La figure 3 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.

Le système **calvin2**+DICE a été développé à l'origine sur processeur SPARC. Il est en cours de portage/adaptation sur processeur Itanium (jeu d'instruction 64 bits d'Intel). 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

Participants : François Bodin, Assia Djabelkhir, Romain Dolbeau, Antony Fraboulet, Pierre Michaud, Olivier Rochecouste, André Seznec, Eric Toullec.

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

Les actions de recherche du projet Caps en architecture de processeurs portent sur les mémoires caches, les mécanismes de lancement des instructions, en particulier prédiction de branchement et ordonnancement du séquencement ainsi que sur les structures de processeur multiflot simultané. Une nouvelle direction de recherche est l'étude de nouveaux compromis performance/complexité sur les processeurs.

6.1.1. Étude des mécanismes de séquencement

Participants : Anthony Fraboulet, Pierre Michaud, André Seznec.

6.1.1.1. Prédicteurs de branchement à historique global

Les performances des microprocesseurs actuels reposent de plus en plus sur les mécanismes de prédiction de branchements dynamiques, et en particulier sur les prédicteurs utilisant la corrélation entre les branchements successifs.

Les études menées dans le projet [13][7] ont été exploitées dans le cadre industriel du processeur Compaq Alpha EV8 [21][2].

6.1.1.2. Prédiction de branchement plusieurs blocs en avance

Les prédicteurs de branchement sont d'autant plus précis que la taille des tables du prédicteur est grande, mais utiliser de grandes tables entraîne un temps de réponse de plusieurs cycles. En 1996, nous avons proposé un mécanisme appelé « multiple-block ahead branch predictor » [14] dans le but de prédire plusieurs branchements en parallèle. Ce mécanisme permet aussi d'implémenter des prédicteurs de branchement pipelinés sur plusieurs cycles.

Nous menons une étude visant à montrer que cette approche permet d'utiliser des prédicteurs de branchement de très grande taille et de très grande précision. En particulier, notre étude vise à combiner des mécanismes pipelinés sur des profondeurs différentes pour prédire la direction du branchement et la cible des branchements directs ou indirects.

6.1.2. Compromis complexité matérielle/performance

Participants : Assia Djabelkhir, Olivier Rochecouste, André Seznec, Eric Toullec.

Les mécanismes matériels mis en œuvre dans les processeurs hautes performances sont devenus au fil des années de plus en plus complexes : exécution spéculative dans le désordre, degré superscalaire de plus en plus élevé, prédiction de branchement, de dépendances mémoire et de valeurs, ... Ceci entraine une grande difficulté de mise en œuvre dans plusieurs directions : pipeline de plus en plus profond, consommation électrique de plus en plus élevée, performances difficilement prévisibles, ...

Nous avons commencé des études visant un meilleur compromis complexité matérielle/performance dans trois directions. En premier lieu, nous avons proposé une nouvelle organisation des architectures superscalaires à clusters appelée architecture WSRS (pour « register Write Specialization, register Read Specialization »)

[2] Cette étude a pu être publiée suite à l'abandon du projet EV8 par Compaq en juin 2001

[23]. Sur un processeur WSRS, un cluster d'unités fonctionnelles n'a accès en écriture et lecture qu'à un sous-ensemble des registres physiques. Ceci permet de réduire de manière sensible la taille du fichier de registres, sa consommation électrique et son temps d'accès, ceci permet aussi de réduire la complexité de la logique de sélection et du réseau de « bypass ».

En second lieu, les techniques aujourd'hui utilisées sur les processeurs superscalaires à usage général peuvent aussi s'appliquer dans le contexte des applications enfouies. Cependant le contexte particulier des applications enfouies permet des compromis différents entre ordonnancement dynamique et ordonnancement statique. Nous commençons une étude sur la mise en œuvre de ces nouveaux compromis (support dans le jeu d'instructions, etc.) dans le cadre des applications enfouies.

6.1.3. Partage des ressources matérielles sur un composant processeur

Participants : Romain Dolbeau, André Seznec.

Parmi les solutions pour exploiter les possibilités d'intégration sur un seul composant, deux solutions concurrentes d'exploitation du parallélisme de processus ont été proposées : l'exécution *multiflot simultané* (ou SMT) [32] et la mise en œuvre de multiprocesseurs sur une seul composant ou *Chip Multiprocessor (CMP)*. Le multiflot simultané offre l'avantage de partager l'ensemble des ressources matérielles entre tous les processus alors qu'un composant CMP est moins complexe à mettre en œuvre qu'un processeur SMT de large degré.

Nous avons proposé une structure de processeur parallèle intermédiaire entre le CMP et le multiflôt simultané : CASH (pour CMP And SMT Hybrid)[24]. Comme le CMP, CASH implémente des cœurs de processeurs séparés, mais partage les ressources matérielles dont la complexité croît simplement de manière linéaire avec le nombre d'unités fonctionnelles (caches d'instructions et de données, unités de chargement d'instruction, prédicteurs de branchement).

6.1.4. Modélisation de la skewed-associativité

Participant : Pierre Michaud.

Les travaux passés sur la skewed-associativité, menés au sein du projet CAPS, ont démontré expérimentalement l'efficacité de cette méthode d'organisation des caches [11]. Les raisons de cette efficacité ne sont cependant pas intuitives. Nous avons introduit un modèle analytique de la skewed-associativité [25] qui confirme les résultats expérimentaux et affine notre compréhension de son efficacité. Le modèle met en évidence la nature statistique de l'efficacité de la skewed-associativité, qui ne dépend pas des caractéristiques des applications. Le modèle montre que l'algorithme de placement mis en œuvre en pratique, qui nécessite plusieurs passes, converge assez rapidement vers un placement optimal. Avec seulement trois bancs, la skewed-associativité simule l'associativité complète pour des ensembles de travail faisant jusqu'à 90% de la taille du cache.

6.1.5. HAVEGE : Génération d'aléa irreproductible

Participant : André Seznec.

De nombreuses applications nécessitent un générateur d'aléa non prévisible. En dehors d'une source physique pouvant générer cet aléa, les solutions logicielles actuelles fournissent un débit de quelques dizaines d'octets par seconde.

La complexité extraordinaire de l'état interne des microprocesseurs superscalaires hautes performances (mémoire cache, exécution dans le désordre, prédicteur de branchement, TLBs, ...) a conduit les constructeurs à fournir aux utilisateurs des mécanismes matériels (compteurs de cycles, ...) permettant de « monitorer » les performances. L'utilisation de ces mécanismes permet d'écrire des programmes dont le résultat est dépendant de l'état interne précis du processeur, or cet état interne n'est pas accessible de l'extérieur. D'autre part, l'état interne du processeur est constamment modifié par tous les événements externes (OS, I/Os, ...) [22][26].

Le but de l'étude est d'explorer l'utilisation de cette complexité et inaccessibilité pour développer des générateurs logiciels d'aléa non prévisibles permettant de délivrer plusieurs millions d'octets par seconde.

Un logiciel appelé HAVEGE (pour HArdware Volatile Entropy Gathering and Expansion) a été développé et porté sur plusieurs plates-formes. Ce logiciel est mis à disposition de la communauté pour test et évaluation (No APP IDDN.FR.001.500017.001.S.P.2001.000.10000).

Cette étude est menée en collaboration avec Nicolas Sendrier du projet CODES de l'INRIA Rocquencourt.

6.2. Environnements pour architectures hautes performances

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

Participants : Ronan Amicel, Laurent Bertaux, Stéphane Bihan, François Bodin, Henri-Pierre Charles, Eric Courtois, Amaury Darsch, Karine Heydemann, Laurent Morin, Antoine Monsifrot, Gilles Pokam, André Seznec, Julien Simonnet, Pierre Villalon.

L'obtention de performances sur les architectures hautes performances nécessite 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.

6.2.1. Aide au portage sur les architectures hautes performances

Participants : Antoine Monsifrot, François Bodin.

L'obtention de code fortement optimisé passe par une étape de réécriture du code source, le « tuning ». Cette étape est essentiellement 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.

Nous avons développé un prototype CAHT qui a pour objectif d'aider au choix des transformations adaptées en s'appuyant sur des expériences d'optimisation de situations similaires répertoriées. Ce prototype vise deux types d'applications : les applications numériques développées en Fortran et les applications embarquées développées en C.

Cette année nous avons exploré l'apport des techniques d'apprentissage automatique pour la mise en œuvre des heuristiques de compilateurs [20] pour décider de l'application d'une transformation de code. La thèse d'Antoine Monsifrot [15] devrait être soutenue le 17 décembre.

Ces travaux font l'objet d'une collaboration ponctuelle avec R. Quiniou (action DREAM) et I.C. Lerman du projet SYMBIOSE.

6.2.2. Compilation itérative

Participants : Karine Heydemann, François Bodin, Henry-Pierre Charles.

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'instructions, 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'instructions) grâce à une optimisation s'accompagne souvent de la dégradation d'une autre (taille du code, pression sur les registres) et des compromis sont ainsi 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 standard 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 [5]. 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 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 cœur de ces travaux.

Cette année les travaux ont porté sur l'analyse du compromis entre le comportement du cache instruction, la taille de code et les performances dans le cas du dépliage de boucle [19] et de l'*inlining* de fonctions tout en permettant la compression des parties de code peu utilisées. Il s'agit d'une part de limiter l'expansion du code lorsque le gain en performance est faible ou de permettre une expansion controlée du code si le gain est significatif et, d'autre part, de diminuer la taille du code par compression, quitte à dégrader les performances lorsque l'impact d'une séquence de code sur le temps d'exécution est faible.

6.2.3. Transformations de code et consommation électrique.

Participants : Gilles Pokam, François Bodin.

Dans les systèmes informatique, l'intérêt pour la réduction de la consommation d'énergie relève principalement de deux constats. Tout d'abord, l'essor rapide du marché des systèmes embarqués et des portables repose sur l'emploi de batteries d'alimentation, il est donc indispensable d'étudier de nouvelles solutions technologiques visant à minimiser le surcoût économique induit par l'utilisation de cette forme d'approvisionnement énergétique (durée de vie, fiabilité, capacité des batteries, etc.). De façon plus générale, on ne peut plus concevoir de nos jours des processeurs rapides sans prendre en compte les problèmes de dissipation d'énergie liés à un haut niveau d'intégration et à l'utilisation de hautes fréquences.

Aussi, afin de produire des systèmes à basse consommation d'énergie, de nombreuses solutions matérielles ont été proposées. Cependant, la consommation en énergie d'un processeur ne dépend pas uniquement de son architecture, mais aussi du code exécuté. En particulier, la consommation d'énergie pour une tâche donnée dépend fortement de l'efficacité du code produit par le compilateur. Dans le cas des architectures VLIW (Very Long Instruction Word), ceci est d'autant plus critique que la gestion du parallélisme d'instructions est confiée au compilateur.

Dans le cadre d'une collaboration avec STMicroelectronics (Centres de Boston et Milan) nous étudions l'impact des transformations de code sur la consommation électrique pour l'architecture LX/ST200 développé par STMicroelectronics et HP. Le modèle de consommation a été défini par l'université de Bologne et Milan. La construction d'*hyperblock* est l'une des transformations de code pouvant accroître la consommation électrique d'une application tout en améliorant les performances (pour la plupart des autres transformations de code la consommation diminue avec l'amélioration de performance).Cette année nous avons défini une nouvelle technique pour le calcul d'*hyperblock* (technique permettant de supprimer des branchements dans les codes) qui prend en compte la gestion de la consommation électrique tout en limitant la dégradation des performances.

6.2.4. Utilisation des instructions multimédia

Participants : Stéphane Bihan, François Bodin, Julien Simonnet.

La majorité des processeurs sont aujourd'hui équipés de supports architecturaux permettant le traitement efficace d'applications multimédia (instructions multimédia par exemple).

L'exploitation automatique des instructions multimédia est une tâche complexe qui se fonde sur les techniques de vectorisation. Cependant dans de nombreux cas il est nécessaire d'avoir, de la part de l'utilisateur, une intervention manuelle qui peut s'avérer fastidieuse. De plus, l'aspect variable de ces instructions d'une plate-forme à l'autre pose également des problèmes de portage de code et de mise en œuvre des outils automatiques.

Des travaux de recherche ont donc été réalisés pour pallier ces carences. 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) [16]. L'utilisation des instructions est proposée en mode automatique ou interactif. Un premier prototype a permis de valider le concept. Ce prototype a été affiné et complété par l'ajout d'un module frontal permettant de traiter une large variété de séquences d'expressions allant des instructions assembleurs aux expressions C, en passant par les structures de contrôle du langage C.

Ce travail s'effectue actuellement dans le cadre du contrat MEDEA+ MESA.

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

Participants : François Bodin, Laurent Morin.

L'extraction d'un maximum de performance du compilateur est conditionnée par un bon usage des paramètres d'optimisation et par une préparation du programme pour le couple processeur/compilateur. Le projet ATLLAS offre une plate-forme regroupant l'ensemble des informations nécessaires : l'analyse approfondie du code source en C ainsi que l'analyse du code assembleur par l'infrastructure SALTO.

Le prototype d'outil mis en œuvre cette année se présente sous la forme d'un serveur disposant de l'ensemble des analyses et des transformations - assembleur et C - et d'un client graphique multi-utilisateur visualisant les deux versions du code et permettant de piloter le serveur.

Au cœur du système se trouve une analyse particulière visant à recouper les deux versions du code c'est à dire à déterminer à quelles parties du code C correspond chacune des instructions assembleurs. Elle se base sur une technique de reconstruction et d'analyse des graphes de contrôle de flot que nous avons mise au point cette année.

Cette étude est menée en partenariat avec Thomson Multimédia.

6.2.6. ALISE : Assembly Level Infrastructure for Software Enhancement

Participants : Laurent Bertaux, François Bodin.

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. Contrairement à l'optimisation dans le cadre de stations de travail, il faut non seulement prendre en compte les performances mais aussi les contraintes de taille de code, de consommation électrique et de temps réel.

Alisé est une infrastructure flexible destinée à la mise en œuvre des techniques d'ordonnancement et d'optimisation de codes assembleurs. Ce travail s'appuie sur les travaux antérieurs autour du système Salto [6] et des compilateurs FlexCC (STMicroelectronics - Central R&D).

Un des concepts de base de cette infrastructure est de permettre le développement de phases d'optimisation de manière indépendante de l'architecture cible. Cette approche a pour but de simplifier la réutilisation et la mise en œuvre des algorithmes. Les techniques d'optimisation sont implémentées sous forme de composants logiciels. Les interfaces entre composants sont fondées sur la technologie XML. Le premier prototype est maintenant opérationnel.

Afin de valider l'approche, en collaboration avec ST, une version d'Alisé pour le DSP MMDSP a été mise en œuvre. Sur la base de cette implémentation nous validons les principes de mises en œuvre. En particulier, nous avons étudié des techniques d'ordonnancement avec des modifications du code assembleur afin de produire des codes plus efficaces.

Ces travaux sont effectués dans le cadre d'une collaboration avec STMicroelectronics (Central R&D - site de Crolles).

6.2.7. Absciss : génération de simulateurs hautes performances de jeux d'instructions

Participants : Ronan Amicel, François Bodin, André Seznec.

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 simulateurs 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.

Nos études montrent qu'ABSCISS permet une vitesse de simulation nettement supérieure à celle d'un simulateur classique. D'autre part, contrairement aux systèmes de simulation compilée existants, notre approche pernet de maîtriser le temps de génération du simulateur [17].

Un prototype d'ABSCISS existe pour deux architectures cibles, le processeur TriMedia de Philips et le processeur LX/ST200 de HP et STMicrolectronics, et son reciblage vers l'architecture PowerPC a également été entrepris.

6.2.8. Le jeu d'instructions IA64 et outils logiciels pour la compilation et l'architecture

Participants : François Bodin, Eric Courtois, Henri-Pierre Charles, Amaury Darsch, André Seznec, Pierre Villalon.

Le jeu d'instructions a un impact considérable sur la conception et l'implémentation de nombreux systèmes informatiques tant au niveau matériel que logiciel, et ceci que ce soit pour les systèmes à usage général ou pour les systèmes enfouis.

Le nouveau jeu d'instructions IA64 d'HP/Intel et les jeux d'instructions du processeur Philips Trimedia et du processeur TI C6xxx sont dits EPIC (Explicitly Parallel Instruction Computing). Les jeux d'instructions EPIC permettent d'exposer directement au compilateur le parallélisme d'instructions présent dans les applications. En particulier, ils permettent de gérer par matériel une partie de l'exécution spéculative.

Dans le cadre de ses activités de recherche en architecture et compilation, le projet CAPS a développé un ensemble d'outils (SALTO, Calvin2, Absciss, ...) orientés vers la recherche en microarchitecture et compilation.

Dans le cadre du contrat EPICEA (voir 7.2) ainsi que de l'accueil d'un ingénieur associé INRIA, nous avons commencé le portage et l'adaptation spécifique de l'ensemble de ces outils à l'architecture IA64 d'Intel/HP en vue de sa mise à disposition de la communauté scientifique. Nous avons d'autre part commencé l'étude d'une architecture de processeur à exécution dans le désordre pour ce jeu d'instruction. En compilation, nous avons étudié des techniques de génération de code, pour des boucles vectorielles, spécifiques pour l'architecture Itanium.

7. Contrats industriels

7.1. MEDEA+ A-502 Architectures pour Systèmes Monopuces à Multi-processeurs (2000-2004)

Participants : Stéphane Bihan, François Bodin, Julien Simonnet.

Pour tirer profit de l'énorme puissance de calcul dont disposera la génération des circuits intégrés en 100 nano-mètres et répondre aux besoins toujours croissants du marché, il est urgent de disposer de méthodes de conception efficaces concernant les architectures de multi-processeurs. Le projet MESA , dans le cadre de Medea+ (http://www.medeaplus.org/) adresse ce problème.

Le rôle du projet CAPS dans ce projet est de fournir des outils d'optimisation configurables pour la programmation de systèmes multi-processeurs enfouis et l'exploitation du parallélisme SIMD des instructions multimédia.

7.2. Projet EPICEA (2001-2004)

Participants : François Bodin, Eric Courtois, Amaury Darsch, André Seznec.

Le projet EPICEA est une collaboration entre l'INRIA projet CAPS, l'université de Versailles-Saint-Quentin, le CEA et Bull financée par le ministère de la recherche. Il s'agit de développer un environnement logiciel de programmation pour les applications scientifiques pour les architectures utilisant le jeu d'instruction IA 64.

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 (cf. 6.2).

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 (cf. 6.2).

7.5. Décomposition de codes embarqués pour systèmes hétérogènes (2002-2005)

Participants : François Bodin, Pascal Terjan.

La thèse de Pascal Terjan (débutée au 1er octobre 2002) est financée dans le cadre d'une convention CIFRE avec la société STMicroelectronics.

Il s'agit d'étudier les techniques logicielles permettant de calculer la décomposition d'un programme C en tâches pouvant être réparties sur les différents composants d'un système enfoui. L'étude analysera le potentiel des techniques définies dans le cadre de la parallélisation automatique et de la reconnaissance de structures de calcul dans les codes.

7.6. Compilation et puissance dissipée (2000-2003)

Participants : François Bodin, Gilles Pokam.

Cette étude fait l'objet d'une convention CIFRE avec la société STMicroelectronics pour le financement de la thèse de Gilles Pokam (cf. 6.2).

7.7. Conventions avec la société Intel

Participants : Eric Toullec, Antony Fraboulet, André Seznec.

Les recherches menées sur l'optimisation des structures de fichier de registres ainsi que sur les mécanismes de séquencement dans les processeurs superscalaires sont soutenues par une donation de la société Intel (Convention 4 01 C 0677 00 31308 06 1 + donation de matérielle).

7.8. Projet de jeune-pousse

Participants : Stéphane Bihan, François Bodin, Romain Dolbeau, Karine Heydeman, Laurent Bertaux, Antoine Monsifrot, Laurent Morin, Ronan Amicel, Julien Simonnet.

Une partie importante de l'activité de l'équipe de recherche a été consacrée à la mise en place de la jeune-pousse CAPS entreprise.

L'objectif de la création de CAPS entreprise (http://www.caps-entreprise.com) est de commercialiser un ensemble de logiciels et services permettant la production de codes optimisés pour les systèmes enfouis. Les logiciels ainsi produits tirent le melleur profit du matériel sur lequel ils s'exécutent. CAPS entreprise permet de réduire les temps de mise au point des logi ciels et le coût des systèmes enfouis haute performance.

CAPS entreprise incubée par Emergys et INRIA-Transfert a été lauréat, en émergence, au concours 2002 du ministère de la recherche pour la création d'entreprise.

8. Actions régionales, nationales et internationales

8.1. Consommation électrique et compilation

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

8.2. ARC HIPSOR

Participant : André Seznec.

Les recherches sur la génération d'aléa irreproductible sont faites dans le cadre d'une ARC INRIA appelé HIPSOR (HIgh Performance SOftware Random number generation) associant le projet CAPS et le projet CODES de l'INRIA Rocquencourt.

8.3. RTP Architecture et Compilation

Participants : Francois Bodin, Pierre Michaud, André Seznec.

Les membres de CAPS participent au RTP CNRS Architecture et Compilation :

- André Seznec est membre du comité de pilotage.
- Pierre Michaud participe à l'action spécifique « Nouvelles technologies et nouveaux paradigmes d'architecture ».
- François Bodin participe à l'action spécifique « Compilation pour les systèmes embarqués ».

9. Diffusion des résultats

9.1. Animation de la communauté scientifique

- A. Seznec a été membre des comités de programme des conférences 29th International Symposium on Computer Architecture (ISCA'29), International Conference on Parallel Processing 2002, 7th International Symposium on High Performance Computer Architecture (HPCA'7), 6th Multithreaded Architecture workshop (MTEAC'6) et du congrès africain de recherche en informatique (CARI'2002).
- F. Bodin est membre du comité de rédaction de la revue TSI.

9.2. Enseignement universitaire

F. Bodin et A. Seznec interviennent dans les cours d'architecture et compilation du DEA informatique, du DIIC et du DESS ISA de l'université de Rennes I.

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 intitulée « Architecture des processeurs hautes performances : panorama et perspective" à la conférence CUIC 2002 à Menton en juin 2002 (Conférence des Utilisateurs Informatique du CEA).
- A. Seznec a présenté une conférence intitulée « Design tradeoffs on the EV8 branch predictor » à l'Université de Delft en mars 2002.

9.4. Divers

- A. Seznec est membre élu de la commission d'évaluation de l'INRIA
- F. Bodin est responsable du DEA d'informatique de l'université de Rennes 1.
- Jacques Lenfant est président de la commission de spécialistes informatique de Rennes 1.
- Jacques Lenfant est membre de l'académie des sciences et des technologies.

10. Bibliographie

Bibliographie de référence

[1] 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.

[2] 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.

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

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

[5] E. ROHOU, F. BODIN, C. EISENBEIS, A. SEZNEC. *Handling Global Constraints in Compiler Strategy.* in « International Journal of Parallel Programming », août, 2000.

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

[7] A. SEZNEC, P. MICHAUD. *De-aliased Hybrid Branch Predictors.* Technical Report, numéro RR-3618, Inria, Institut National de Recherche en Informatique et en Automatique, 1999.

[8] F. BODIN, P. BECKMAN, D. GANNON, J. SRINIVAS. *Sage++ : a class library for building Fortran and C++ restructuring tools.* in « Proceedings of the Second Object-Oriented Numerics Conference », avril, 1994.

[9] F. BODIN, W. JALBY, C. EISENBEIS, D. WINDHEISER. *Window-based register allocation.* in « Code Generation - Concepts, Tools, Techniques, Proceedings of the International Workshop on Code Generation », 1991, pages 119-145.

[10] F. BODIN, L. KERVELLA, T. PRIOL. *Fortran-S : a fortran interface for shared virtual memory architectures.* in « Proceedings of Supercomputing », éditeurs I. C. S. PRESS., pages 274-283, novembre, 1993.

[11] F. BODIN, A. SEZNEC. *Skewed associativity improves performance and enhances predictability.* in « IEEE Transactions on Computers », mai, 1997.

[12] C. EISENBEIS, W. JALBY, D. WINDHEISER, F. BODIN. *A strategy for array management in local memory.* in « Journal of Mathematical Programming », numéro 63, 1994, pages 331-370.

[13] 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 », éditeurs IEEE-ACM., Denver, juin, 1997.

[14] 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 recherche

[15] A. MONSIFROT. *Utilisation du raisonnement à partir de cas et de l'apprentissage pour l'optimisation de code.* thèse de doctorat, décembre, 2002.

Articles et chapitres de livre

[16] G. POKAM, S. BIHAN, J. SIMONNET, F. BODIN. *SWARP : A Retargetable Preprocessor for Multimedia Instructions.* in « Concurrency and Computation : Practice and Experience », A paraître.

Communications à des congrès, colloques, etc.

[17] R. AMICEL, F. BODIN. *Mastering Startup Costs in Assembler-Based Compiled Instruction-Set Simulation.* in « 6th Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT-6) », Cambridge, États-Unis, Février, 2002.

[18] R. ESPASA, F. ARDANAZ, J. EMER, S. FELIX, J. GAGO, R. GRAMUNT, I. HERNANDEZ, T. JUAN, G. LOWNEY, M. MATTINA, A. SEZNEC. *Tarantula : A Vector Extension to the Alpha Architecture.* in « Proceedings of the 29th International Symposium on Computer Architecture (IEEE-ACM) », Anchorage, mai, 2002.

[19] K. HEYDEMANN, F. BODIN, P.M.W. KNIJNENBURG, L. MORIN. *UFC : a Global Trade-off Strategy for Loop Unrolling for VLIW Architecure..* in « Proceedings of the 10th Workshop on Compilers for Parallel Computers », to appear.

[20] A. MONSIFROT, F. BODIN. *A Machine Learning Approach to Automatic Production of Compiler Heuristics.* in « Tenth International Conference on Artificial Intelligence : Methodology, Systems, Applications (AIMSA 2002) », septembre, 2002.

[21] A. SEZNEC, S. FELIX, V. KRISHNAN, Y. SAZEIDES. *Design trade-offs on the EV8 branch predictor.* in « Proceedings of the 29th International Symposium on Computer Architecture (IEEE-ACM) », Anchorage, mai, 2002.

[22] A. SEZNEC, N. SENDRIER. *HAVEGE : Hardware volative entropy gathering and expansion unpredictable random number generation at user level.* in « Workshop on Random Number Generators and Highly Uniform Point Sets », Montréal, June, 2002.

[23] A. SEZNEC, E. TOULLEC, O. ROCHECOUSTE. *Register Write Specialization Register Read Specialization : A Path to Complexity Effective of Wide Issue Superscalar Processors.* in « Proceedings of the 35th International Symposium on Microarchitecture (IEEE-ACM) », Istamboul, novembre, 2002.

Rapports de recherche et publications internes

[24] R. DOLBEAU, A. SEZNEC. *CASH : Revisiting hardware sharing in single-chip parallel processor.* Rapport de recherche, numéro 1491, IRISA, octobre, 2002.

[25] P. MICHAUD. *A Statistical Study of Skewed Associativity.* Rapport de recherche, numéro 1489, IRISA, octobre, 2002.

[26] A. SEZNEC, N. SENDRIER. *HArdware Volatile Entropy Gathering and Expansion : generating unpredictable random number at user level.* Rapport de recherche, numéro 1492, IRISA, octobre, 2002.

Bibliographie générale

[27] D. BACON, S. GRAHAM, O. SHARP. *Compiler Transformations for High-Performance Computing.* in « ACM Computing Surveys », numéro 4, volume 26, décembre, 1994, pages 345-420.

[28] P. CHANG, E. HAO, Y. PATT. *Target prediction for indirect jumps.* in « Proceedings of the 24th Annual International Symposium on Computer Architecture », 1997.

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

[30] M. ROSEMBLUM, S. HERROD, E. WITCHEL, A. GUPTA. *Complete computer system simulation : the SimOS approach.* in « IEEE Parallel and Distributed Technology », volume n° 3, 1995.

[31] 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.

[32] D. TULLSEN, S. EGGERS, H. LEVY. *Simultaneous multithreading : maximising on-chip parallelism.* in « 22nd Annual International Symposium on Computer Architecture », pages 392-403, juin, 1995.

[33] R. UHLIG, T. MUDGE. *Trace-Driven memory simulation : a survey.* in « ACM Computing Surveys », 1997.

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