Prédiction de l'adresse des lectures pour tolérer la latence des accès mémoire

Pham Tuong Hai, Christine Rochange, Pascal Sainrat, Daniel Litaize[a]

[a] Université Paul Sabatier, Institut de Recherche en Informatique de Toulouse
118 route de Narbonne, 31062 Toulouse cedex 04, France
{auteur}@irit.fr

Résumé

Un des points clés de l'amélioration de la performance des processeurs superscalaires est la tolérance de la latence des accès aux données en mémoire. Nous proposons de résoudre ce problème en prédisant, lors de l'émission dans une station de réservation d'une instruction de lecture, l'adresse à laquelle elle va accéder. Ainsi, les instructions de lecture peuvent être exécutées plus tôt, masquant ainsi tout ou partie de la latence de la hiérarchie mémoire. La stratégie de prédiction de l'adresse ressemble à celle utilisée par les mécanismes de préchargement mais sert ici non pas à charger un bloc dans le cache mais à charger une valeur dans le processeur. Cette valeur ne sera utilisée par d'autres instructions que lorsque la prédiction aura été vérifiée, c'est à dire lorsque l'adresse de la lecture aura été réellement calculée. Ce procédé permet d'améliorer la performance du processeur de manière sensible.

Mots-clés : Architecture, processeurs superscalaires, préchargement de données

1. Introduction

Les processeurs superscalaires à exécution non ordonnée tirent parti de la possibilité d'exécuter les instructions dans un ordre différent de celui du programme pour masquer la latence de certains accès à la mémoire par l'exécution d'autres instructions. Ce n'est toutefois pas suffisant pour masquer la latence de certaines lectures qui nécessitent un accès au cache de deuxième niveau ou –pire– à la mémoire principale [2].

Afin de résoudre ce problème, nous proposons de prédire les adresses auxquelles vont accéder les instructions de lecture afin de pouvoir les exécuter plus tôt.

En premier lieu, nous présentons notre modèle d'exécution ainsi que les programmes tests, puis nous explicitons le mécanisme de prédiction d'adresse et présentons ses résultats en termes de qualité de prédiction. Nous montrons ensuite le gain en performance apporté par notre dispositif sur la performance du processeur en termes de nombre d'instructions exécutées par cycle (IPC).

2. Modèle d'exécution - Programmes de test

L'architecture modélisée met en œuvre une politique d'exécution des instructions non ordonnée et spéculative avec gestion précise des exceptions. Le nombre maximum d'instructions pouvant être émises par cycle est de 8 et 96 instructions peuvent être en cours dans le processeur, valeur optimale définie dans [7]. La configuration des unités fonctionnelles et de leurs stations de réservation est elle-aussi optimale selon les résultats exposés dans [6]. Les instructions choisies dans les stations pour être exécutées sont les plus anciennes parmi celles dont les opérandes sont calculés. On suppose que les branchements sont toujours correctement prédits. En ce qui concerne les lectures et les écritures, nous avons modélisé le mécanisme de détection des synonymes proposé dans [3] qui permet d'effectuer une lecture avant une écriture même si l'adresse de celle-ci n'est pas connue en prédisant qu'il ne s'agit pas d'un synonyme.

La hiérarchie mémoire est composée de deux niveaux de caches. Le cache de données de premier niveau est de 32 Kilo-octets. Il est à correspondance directe, à réécriture et à écriture allouée avec chargement sur écriture [5]. Il supporte quatre accès simultanés depuis le processeur grâce à un entrelaçage sur quatre bancs et est non bloquant [4]. En cas de succès de l'accès dans ce cache, le temps d'accès est de 1 cycle.

Le cache de deuxième niveau est un cache à écriture différée mono-port, non bloquant. Sa taille est de 256 Kilo-octets (taille volontairement limitée car les programmes utilisés ont des ensembles de données petits). Il est associatif par ensembles de 4 blocs. Le temps d'accès à ce cache est de 4 cycles en cas de succès.

La mémoire principale se divise en 8 bancs de 1 Méga-octets. Son temps d'accès représente 20 cycles du processeur. L'interface entre le cache de deuxième niveau et la mémoire principale est un bus à transactions éclatées dont le débit est de 64 bits/cycle.

Le cache d'instructions de premier niveau est supposé infini, ce qui influe peu sur les résultats car les programmes utilisés sont petits et seraient contenus dans un cache de 32 Kilo-octets.

Les programmes utilisés sont issus des SPEC92. Afin de pouvoir évaluer l'influence de la hiérarchie mémoire, nous n'avons retenus qu les 8 qui utilisent le plus grand nombre de données : *espresso, compress, tomcatv, alvinn, ear, swm256, su2cor, hydro2d.*

Le simulateur que nous avons développé est dirigé par programme et exécute du code objet Motorola 88k produit par le compilateur mcc88k de Microtec Research avec l'optimiseur actif. Les librairies ont été partiellement réécrites excepté les appels au système opératoire qui sont émulés et ne sont donc pas exécutés par le processeur.

3. Le mécanisme de prédiction d'adresse

Les lectures peuvent être divisées en 3 catégories :

1. les lectures à *pas nul*. Elles accèdent toujours à la même adresse.
2. Les lectures à *pas constant*. Elles accèdent en mémoire à des adresses incrémentées (ou décrémentées) régulièrement.
3. Les lectures à *pas irrégulier*.

Comme la prédiction de branchement, la prédiction d'adresse déduit le comportement des instructions à partir des exécutions précédentes. A chaque cycle, huit instructions sont chargées et donc jusqu'à huit instructions peuvent être prédites en parallèle.

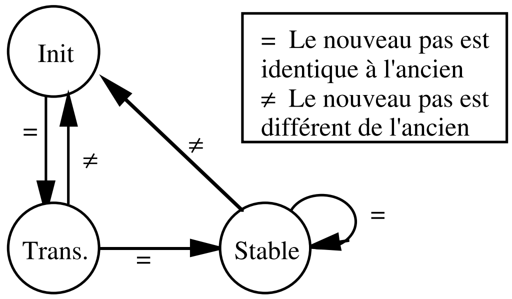
Si une instruction de lecture a été prédite, elle peut être exécutée dès qu'elle est parvenue dans une station de réservation. Lorsqu'elle est sélectionnée, si ses opérandes sont prêts, le calcul d'adresse est effectué sinon l'accès se fait à l'adresse prédite. Dans ce dernier cas, la donnée renvoyée par le cache est mémorisée dans la station de réservation. L'instruction est ré-exécutée lorsque ses opérandes sont prêts afin de calculer l'adresse. Si l'adresse prédite est correcte, la donnée est diffusée aux stations qui en ont besoin sinon la lecture est relancée avec l'adresse correcte.

Nous avons considéré deux cas extrêmes. Le cas *parfait* prédit correctement toutes les instructions de lecture. Ce cas nous permet d'évaluer l'accroissement de performance maximum (en nombre d'instructions exécutées par cycle) que peut apporter la prédiction d'adresse à un processeur de type similaire à celui que nous avons modélisé.

Le cas *réel* étudié est un modèle relativement peu coûteux matériellement. Le mécanisme de prédiction des adresses est largement inspiré de [1]. Il comporte une table de 256 entrées (taille permettant d'obtenir une bonne qualité de prédiction pour les programmes considérés) qui est accédée en correspondance directe avec l'adresse de l'instruction à prédire. Cette table est chargée de mémoriser le comportement des lectures. Pour ce faire, chaque entrée est composée de 5 champs : le champ **adresse_instruction** permet de vérifier que les informations de l'entrée concerne bien l'instruction considérée. Le champ **adresse_accès** garde l'adresse du dernier accès effectué par cette instruction. Le champ **pas** est la différence entre les adresses des deux derniers accès de cette instruction. Le champ **état** indique si l'on peut effectuer une prédiction ou pas. Le champ **avance** indique le nombre d'occurrences de cette instruction présentes dans le processeur qui ont été prédites et qui n'ont pas encore mis à jour la table.

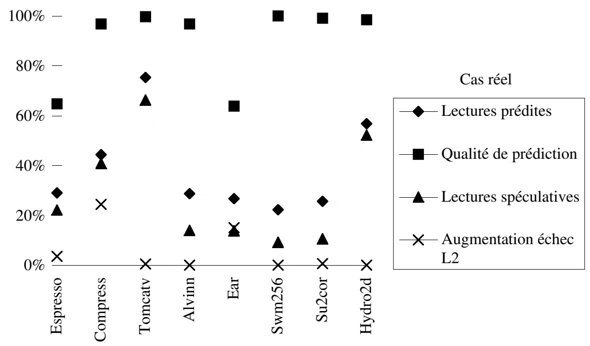
Avant l'émission d'une instruction dans une station de réservation, si l'entrée contient des données relatives à cette instruction (c'est à dire si le champ **adresse_instruction** est égal à l'adresse de l'instruction), et si la prédiction est autorisée (état *Stable*), l'**avance** est incrémentée et l'adresse prédite est $adresse\_acc\grave{e}s + (pas \times avance)$. Cette adresse prédite est placée dans la station de réservation en même temps que l'instruction.

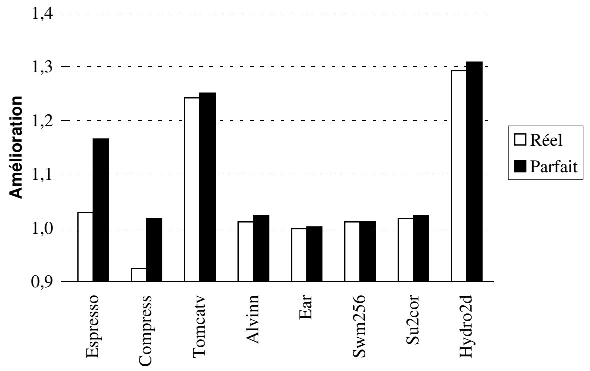
Lors de la mise à jour, si l'instruction n'est pas présente dans la table, elle y est placée avec **état**=*Init*, **avance**=1 et **pas**=0. Sinon, le champ **pas** est mis à jour avec la différence entre l'adresse présente dans la table et celle qui est accédée, le champ **état** évolue suivant l'automate à états ci-contre, le champ **avance** est décrémenté s'il est différent de 0 et l'adresse accédée est placée dans le champ **adresse_accès**.



Deux méthodes de mise à jour ont été évaluées. Soit la table est mise à jour dans l'ordre du programme c'est à dire lorsque les instructions sont retirées du tampon de réordonnancement. La mise à jour est alors assez tardive et la prédiction est fortement anticipée par rapport à l'information présente dans la table car les boucles ne sont généralement pas longues et plusieurs itérations d'une boucle peuvent être présentes dans le processeur. Soit la table est mise à jour le plus tôt possible, c'est à dire dès que l'adresse de l'accès est calculée. La table est alors mise à jour dans le désordre, ce qui peut nuire à la qualité de la prédiction. Toutefois, la politique d'amorçage des instructions est d'exécuter l'instruction la plus ancienne, ce qui favorise l'exécution dans l'ordre. Il n'est donc pas étonnant que nous n'ayons pas observé, en moyenne, de différence notable entre ces deux méthodes. Les résultats ci-dessous sont donnés pour une mise à jour dans l'ordre.

4. Résultats





La moyenne des améliorations mesurées sur l'ensemble des 8 programmes est supérieure à 5%

dans le cas *réel* et atteint presque 10% pour une prédiction *parfaite*. Le gain de performance est donc très sensible si l'on considère que le processeur modélisé sans notre dispositif a déjà un IPC moyen proche de 2.

Compress voit son IPC diminué quand notre système de prédiction réel est actif. Ceci provient d'une augmentation significative du nombre d'échecs dans le cache de deuxième niveau. Toutefois, cette augmentation disparaît avec l'amélioration de la prédiction. D'autres programmes profitent modérément de notre système car moins de 20 % des lectures sont exécutées de manière spéculative (les opérandes sont souvent disponibles au moment où la lecture est lancée). Enfin, *tomcatv* et *hydro2d* présentent une excellente amélioration de performance car plus de 50% des lectures sont prédites et exécutées spéculativement et génèrent peu d'échecs supplémentaires au niveau des caches.

5. Conclusion et perspectives

Ce mécanisme diffère des mécanismes classiques de préchargement sur plusieurs points.

Les mécanismes de préchargement se contentent de charger un bloc dans le cache de premier niveau alors que notre dispositif de prédiction d'adresse charge directement une donnée dans le processeur. C'est un point important car la bande passante entre les caches de premier et second niveau est limitée et constitue un goulet d'étranglement [8].

Le préchargement engendre plus de requêtes inutiles car il prédit à la fois l'occurence d'une instruction de lecture et son adresse. Dans notre cas, on est certain qu'une instruction de lecture va être exécutée. Les requêtes au cache sont donc toujours justifiées (hormis en cas de mauvaises prédictions de branchement), seule la prédiction de l'adresse peut être erronée.

Le préchargement charge un bloc pour la prochaine occurence de l'instruction. A priori, il lance donc une requête plus tôt que notre système. Mais le plus souvent, deux occurences successives d'une instruction de lecture sont présentes dans le processeur et ont donc également été prédites par notre système.

La prédiction d'adresse parfaite montre que cette approche permet d'améliorer les performances du processeur d'environ 10%. Le cas simple étudié ici apporte déjà une amélioration notable (plus de 5%). Il reste de nombreuses alternatives à étudier et à évaluer. Par exemple, des mécanismes peuvent sans doute être mis en œuvre pour limiter l'augmentation de l'occupation du cache de niveau 2.

Bibliographie

1. J.-L. Baer et T.-F. Chen, *An Effective On-Chip Preloading Scheme To Reduce Data Access Penalty*, Supercomputing'91, novembre 1991.
2. D. Burger, J. R. Goodman et A. Kägi, *Quantifying Memory Bandwidth Limitations of Current and Future Microprocessors*, 23ème International Symposium on Computer Architecture, mai 1996.
3. M. Franklin, *The Multiscalar Architecture*, PhD thesis, University of Wisconsin, Madison, 1993.
4. D. Kroft, *Lockup-Free Instruction Fetch/Prefetch Cache Organization*, 8ème Annual International Symposium on Computer Architecture, mai 1981.
5. N. P. Jouppi, *Cache Write Policies and Performance*, 20ème Annual International Symposium on Computer Architecture, juin 1993.
6. S. Jourdan, P. Sainrat et D. Litaize, *Exploring Configurations of Functional Units in an Out-Of-Order Superscalar Processor*, 25ème International Symposium on Computer Architecture, juin 1995.
7. S. Jourdan, P. Sainrat et D. Litaize, *Tampons d'amorçage des microprocesseurs superscalaires à exécution non ordonnée : une analyse*, Renpar'8, mai 1996.
8. A. Seznec et F. Lloansi, *About Effective Cache Miss Penalty on Out-Of-Order Superscalar Processors*, rapport recherche INRIA 2726, novembre 1995.