Optimisations du chargement des instructions

Thierry Haquin, Philippe Reynes, Christine Rochange, Pascal Sainrat

RENPAR'14 / ASF / SYMPA
Hamamet, Tunisie, 10 - 13 avril 2002

Optimisations du chargement des instructions

Thierry Haquin, Philippe Reynes, Christine Rochange et Pascal Sainrat

Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier, 118 route de Narbonne
31062 Toulouse cedex 4 - France
haquin@irit.fr

RÉSUMÉ: *Les processeurs actuels et à venir, dont le cœur d'exécution exploite le parallélisme entre instructions, ne peuvent atteindre leurs performances maximales que s'ils sont alimentés par un débit d'instructions suffisant. Dans cet article, nous montrons que la bande passante d'accès au cache d'instructions est en général sous-exploitée. Nous proposons deux solutions pour optimiser les accès au cache d'instructions : l'une consiste à combiner plusieurs accès à une même ligne de cache ; l'autre prévoit de réordonner les accès pour limiter le nombre de conflits de bancs dans un cache multi-port. Les résultats de simulation montrent que ces deux optimisations améliorent sensiblement le débit de chargement des instructions. Par ailleurs, leur mise en oeuvre se fait au travers de séquences de contrôle du chargement qui tiennent également lieu de prédicteur multiple de branchements.*

MOTS-CLÉS : *chargement des instructions, cache d'instructions, prédiction multiple de branchements*

1. Introduction

Les performances potentielles d'un cœur d'exécution superscalaire ne peuvent être atteintes que si le débit de chargement des instructions est suffisant. Or, la bande passante de chargement est généralement sous-exploitée à cause (a) du découplage entre ce que le processeur veut lire (blocs de base) et ce que le cache fournit (lignes de cache) et (b) des conflits de bancs entre les adresses soumises aux différents ports du cache. Nous proposons deux techniques pour améliorer l'utilisation de la bande passante disponible. La première consiste à éliminer des accès redondants au cache (accès à une même ligne, quand, par exemple, deux blocs de base proches dans le flot d'exécution sont situés dans cette même ligne) ; la seconde réordonne les accès au cache sur plusieurs cycles, de manière à réduire le nombre de conflits de bancs. La mise en œuvre de ces techniques passe par la construction de séquences de contrôle du chargement qui peuvent, en outre, être utilisées pour prédire plusieurs branchements par cycle. Après discussion des résultats de simulation préliminaires, nous suggérons une implémentation matérielle commune pour ces deux techniques et évaluons ses performances.

2. Chargement des instructions : état de l'art

Comme nous l'avons dit, le cœur d'un processeur superscalaire doit, pour atteindre ses performances maximales, être suffisamment alimenté en instructions. Les solutions proposées dans la littérature pour améliorer le débit de chargement des instructions sont les suivantes :

- mise en œuvre d'un *prédicteur de branchement* : ce composant a été largement étudié par le passé. Son rôle est d'anticiper le résultat des opérations de contrôle de flot afin d'éviter de charger des instructions sur un chemin incorrect.
- réduction du *taux de défauts dans le cache d'instructions*, grâce à des organisations de cache adaptées et à la mise en œuvre de techniques de préchargement.

- utilisation d'un *cache d'instructions multi-port* : avec un cache monoport, un bloc de base ne peut pas toujours être chargé en un seul cycle, bien que sa taille (de l'ordre de 5 instructions en moyenne) soit en général inférieure à celle d'une ligne de cache (souvent 8 ou 16 instructions). C'est le cas lorsqu'il est à cheval sur deux lignes de cache : il est alors chargé en deux accès, et donc en deux cycles. Avec un cache bi-port constitué de deux bancs entrelacés et un bus de largeur suffisante pour transférer deux lignes de cache, ce même bloc peut être chargé en un seul cycle (les deux lignes de cache nécessaires sont consécutives en mémoire, donc rangées chacune dans un des deux bancs, et peuvent ainsi être lues en parallèle).
- mise en œuvre d'un mécanisme de *chargement de plusieurs blocs de base en un seul cycle* : pour cela, il faut disposer d'un cache multi-port et d'un prédicteur muliple de branchements qui génère, à chaque cycle, plusieurs adresses de blocs de base pas forcément consécutifs dans l'espace d'adressage. Ces dernières années, de nombreuses études ont été menées sur la prédiction multiple de branchements. Certains prédicteurs multiples [2][4][5][6] génèrent les adresses de blocs de base au moment du chargement, à partir d'une table des adresses cibles (BTB) et de l'historique des branchements : cette solution est complexe, même si on utilise des techniques de pipeline. D'autres [3] se basent sur des traces enregistrées à l'exécution (hors du chemin critique) et prédisent que le programme suivra le même chemin que par le passé.

3. Comment optimiser les accès au cache d'instructions ?

3.1 Principaux problèmes et solutions proposées

Nous avons vu qu'un cache multi-port permettait le chargement simultané de plusieurs lignes (et donc, de plusieurs blocs de base ou d'un bloc de base à cheval sur plusieurs lignes). Toutefois, si ce cache est organisé en plusieurs bancs entrelacés, le débit de chargement des instructions peut être limité par les conflits entre bancs : si plusieurs adresses soumises au cache correspondent à un même banc, elles ne peuvent être lues au même cycle. Le risque de conflits peut être réduit en augmentant le nombre de bancs, mais cela a tendance à accroître le temps d'accès. Dans cet article, nous proposons une nouvelle approche qui consiste à accéder au cache d'instructions dans le désordre.

Une autre amélioration possible consiste en l'élimination des accès redondants : quand deux blocs de base proches dans le flot dynamique d'instructions sont situés dans une même ligne de cache, cette ligne est normalement lue deux fois. De même, quand un bloc de base est exécuté plusieurs fois de suite dans une petite boucle, la même ligne est lue à chaque itération. Nous verrons comment il est possible de combiner plusieurs accès à une même ligne de cache.

La figure 1 présente un scénario que nous utiliserons pour illustrer le fonctionnement de notre dispositif. Une séquence dynamique de blocs de base (A-B-C-A-B-C-D) est exécutée (sur le schéma, A' représente la seconde occurence du bloc A dans le flot d'exécution). Le tableau indique comment les blocs sont rangés dans le cache d'instructions (numéro de ligne, position du bloc dans la ligne, numéro de banc). Il donne également les positions des blocs dans la trace. Notons que les blocs B et C appartiennent à la même ligne, tandis que le bloc D est à cheval sur deux lignes (il est décomposé en deux parties, D_1 et D_2).

La figure 2 indique le nombre de cycles nécessaires pour charger les 7 blocs de la séquence, en considérant un cache parfait (taux de défauts nul), avec 2 ports et 4 bancs. Le mécanisme de base (sans réordonnancement ni combinaison des accès) nécessite 7 cycles : la bande passante disponible (2 lignes par cycle) est sous-exploitée à cause des conflits de bancs entre les lignes ℓ_0 et ℓ_1. La colonne suivante montre comment le réordonnancement des accès peut réduire la durée de chargement d'un cycle, puisque le bloc D peut être lu pendant les deux premiers cycles, en parallèle avec les blocs A et B. La combinaison des accès utilisée seule (sans réordonnancement) réduit le nombre d'accès : une seule lecture de la ligne ℓ_1 permet de charger les blocs B et C. Notons que comme il n'y a pas de

réordonnancement, la seconde occurrence du bloc A nécessite un second accès à la ligne ℓ_0 (sinon A' serait lu avant B et C). Enfin, la dernière colonne montre que lorsque l'on associe le réordonnancement et la combinaison des accès, le temps de chargement de la séquence n'est que de 2 cycles, avec un seul accès à chacune des 4 lignes.

Bloc de base	A	B	C	A′	B′	C′	D
Ligne de cache	ℓ_0	ℓ_1	ℓ_1	ℓ_0	ℓ_1	ℓ_1	ℓ_2 | ℓ_3
Instructions dans la ligne	2 .. 4	1 .. 5	6 .. 7	2 .. 4	1 .. 5	6 .. 7	3..7 | 0..2
Banc du cache	0	0	0	0	0	0	1 | 2
Position dans la trace	0	3	8	10	13	18	20 | 24

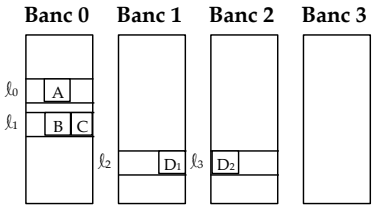


FIGURE 1 : SCENARIO-EXEMPLE

	Base		**Réordonnancement**		**Combinaison**		**Réord. & Combin.**	
CYCLE	**PORT 0**	**PORT 1**	**PORT 0**	**PORT 1**	**PORT 0**	**PORT 1**	**PORT 0**	**PORT 1**
1	ℓ_0 (A)		ℓ_0 (A)	ℓ_2 (D_1)	ℓ_0 (A)		ℓ_0 (A,A′)	ℓ_2 (D_1)
2	ℓ_1 (B)		ℓ_1 (B)	ℓ_3 (D_2)	ℓ_1 (B,C)		ℓ_1 (B,C,B′,C′)	ℓ_3 (D_2)
3	ℓ_1 (C)		ℓ_1 (C)		ℓ_0 (A′)			
4	ℓ_0 (A′)		ℓ_0 (A′)		ℓ_1 (B′,C′)	ℓ_2 (D_1)		
5	ℓ_1 (B′)		ℓ_1 (B′)		ℓ_3 (D_2)			
6	ℓ_1 (C′)	ℓ_2 (D_1)	ℓ_1 (C′)					
7	ℓ_3 (D_2)							

FIGURE 2 : REDUCTION DU TEMPS DE CHARGEMENT PAR REORDONNANCEMENT ET COMBINAISON

Nous avons évalué les performances potentielles (en termes de réduction du temps de chargement) des deux techniques que nous proposons, en dehors de toute considération sur leur implémentation matérielle. Pour ce faire, nous avons modélisé un processeur frontal muni d'un prédicteur de branchements parfait et d'un cache d'instructions multi-port, sans défauts, avec des lignes de 8 instructions. Chacune des deux techniques est appliquée sur une fenêtre *non-glissante* de 3 cycles : tous les 3 cycles, on calcule la séquence des adresses qui vont être soumises au cache pendant les 3 cycles à venir (si un bloc est à cheval sur plusieurs lignes de cache, il doit être complètement chargé dans une même fenêtre). Le nombre de blocs de base (ou de parties de blocs) qui peuvent être chargées en un même cycle est limité à deux fois le nombre de ports (ceci pour éviter d'avoir des séquences irréalistement longues, par exemple dans le cas de boucles). Dans notre exemple, cela signifie que la ligne ℓ_3 ne doit pas être lue en même temps que la ligne ℓ_1, mais au cycle suivant.

Nous avons simulé l'exécution des 300 premiers millions d'instructions de 8 programmes entiers issus de la suite SPEC2000 (*gcc*, *vpr*, *eon*, *parser*, *vortex*, *gzip*, *twolf* et *bzip2*). Les courbes qui suivent représentent la moyenne des résultats obtenus pour ces 8 benchmarks.

La figure 3 montre la réduction du temps de chargement en fonction du nombre de bancs dans le cache. Les résultats montrent que la combinaison des accès améliore sérieusement les performances : le temps de chargement est divisé par 2,8 avec un cache bi-port et par 5,8 avec un cache quadri-port. Ceci confirme que, avec le mécanisme de chargement de base, le nombre d'accès redondants est important et engendre une consommation inutile de la bande passante. Lorsque le réordonnancement est appliqué seul, il a un impact limité mais il apporte un gain plus sensible lorsqu'il est associé à la combinaison des accès. Ceci est dû au fait qu'en l'absence de combinaison, les accès redondants génèrent trop de conflits de bancs qui ne peuvent être tous résolus par réordonnancement. La combinaison, en supprimant les accès redondants et les conflits de bancs associés, laisse plus de liberté pour résoudre les autres conflits par réordonnancement. Notons que le gain est d'autant plus important que le nombre de ports de cache est grand : ceci est lié au fait que le nombre de blocs qui

peuvent être chargés en un même cycle a été limité à deux fois le nombre de ports. Ainsi, quand un bloc est exécuté un grand nombre de fois dans une petite boucle, un seul accès à sa ligne de cache permet d'alimenter le cœur d'exécution pour 4 itérations avec un cache bi-port, et pour 8 itérations avec un cache quadri-port. Les courbes montrent également que 4 bancs dans le cache sont suffisants pour obtenir la majeure partie du gain.

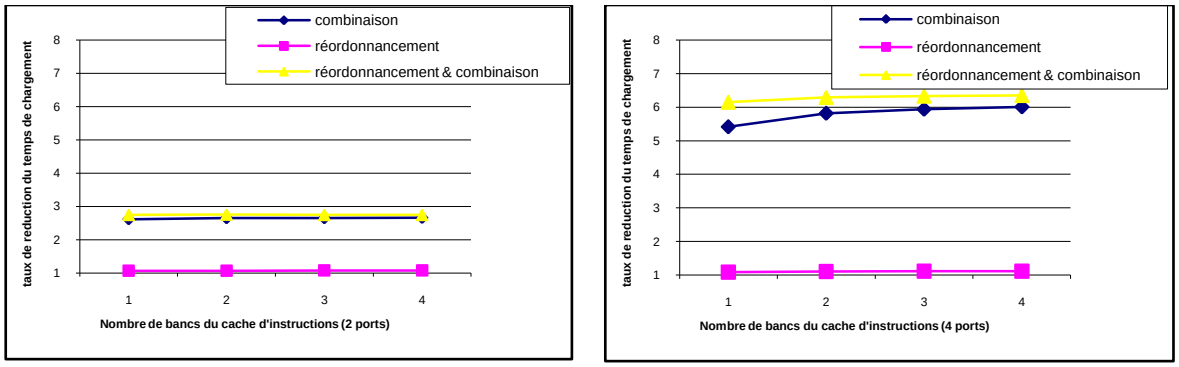


FIGURE 3 : REDUCTION DU TEMPS DE CHARGEMENT PAR REORDONNANCEMENT ET COMBINAISON DES ACCES

Nous avons également mesuré que le réordonnancement (seul ou associé à la combinaison) est d'autant plus intéressant que la fenêtre de chargement est large. Toutefois, dans l'objectif d'une implémentation matérielle des deux techniques, il est probablement souhaitable de maintenir la fenêtre aussi courte que possible.

3.2 Mise en œuvre matérielle des solutions proposées

Le mécanisme que nous proposons (représenté sur la figure 4) repose sur la construction et l'exploitation de *séquences de contrôle du chargement*, suites d'adresses de lignes de cache à charger successivement. Ces séquences sont mémorisées dans une *table de séquences*.

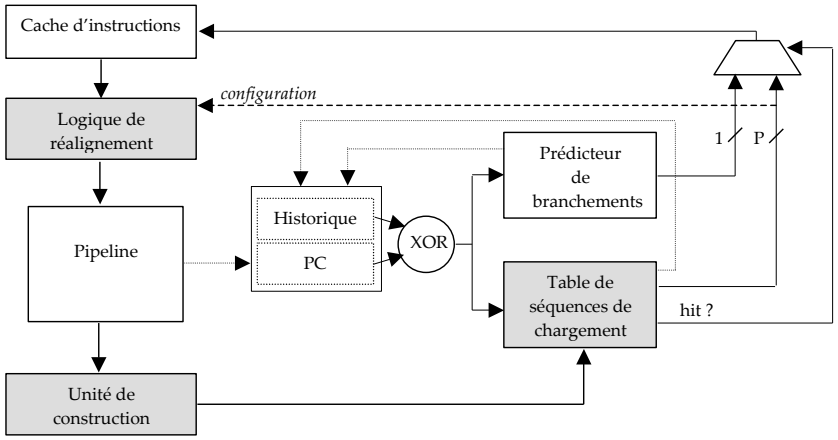


FIGURE 4 : VUE GENERALE DE NOTRE MECANISME

Dans l'étage de chargement, le PC et l'historique des branchements sont utilisés pour accéder en parallèle à la table de séquences et au prédicteur de branchements. L'accès au cache d'instructions se fait alors à partir d'une séquence de contrôle trouvée dans la table ou à partir de l'adresse du prochain bloc de base fournie par le prédicteur de branchements si aucune séquence n'a été trouvée. Ainsi, lorsqu'une séquence de contrôle est trouvée dans la table, elle tient lieu de prédiction de branchement (prédiction de type "1 bit" puisqu'elle prévoit le chargement des blocs de base qui ont précédemment

été exécutés les uns à la suite des autres). Nous verrons dans le paragraphe 4 que cette prédiction est suffisamment fiable pour être retenue à la place d'un prédicteur multiple.

La construction d'une *séquence de contrôle* passe par une analyse du flot d'instructions (suite dynamique de blocs de base) pour : (a) calculer les adresses des lignes de caches qui contiennent ces blocs de base ; (b) détecter les adresses dupliquées et combiner les requêtes correspondantes ; (c) réordonner les requêtes pour éviter les conflits de bancs entre adresses qui seront soumises simultanément au cache.

Une séquence de contrôle est composée de plusieurs *commandes de chargement*, dont chacune contient des adresses de lignes à soumettre en parallèle au cache d'instructions et des informations de routage nécessaires pour insérer les instructions lues dans la file d'instructions dans l'ordre dicté par le programme. La figure 5 montre la structure de la séquence de contrôle correspondant à l'exemple de la figure 1 (3 commandes, puisque notre contrainte sur le nombre de blocs chargés en un cycle empêche la lecture simultanée de ℓ_1 et ℓ_3). Les informations de routage sont organisées en *cellules* dont chacune représente soit un bloc de base entier, soit une partie de bloc de base adjacente à une frontière de ligne de cache. Elle inclut le numéro du port qui doit renvoyer la ligne de cache contenant le bloc, un *masque de sélection* qui désigne les instructions de la ligne appartenant au bloc, et la *position* de la première instruction du bloc de base dans la trace. Les informations de routage sont exploitées par la *logique de réalignement* (illustrée sur la figure 6) : tout d'abord, les instructions associées à une commande de chargement sont extraites des lignes de cache lues, en fonction du numéro de port et du masque de sélection. Elles sont ensuite décalées vers leur place correcte dans la file des instructions en fonction de leur position dans la trace. Notons qu'il y a autant de décaleurs que de cellules (parties de bloc) dans une commande : les sorties de tous les décaleurs sont ensuite combinées par un OU logique pour construire une trace partielle qui est rangée dans la file d'instructions. Les instructions peuvent être décodées et amorcées aussitôt qu'elles entrent dans la file d'instructions, à la condition que toutes les instructions précédentes aient déjà été chargées.

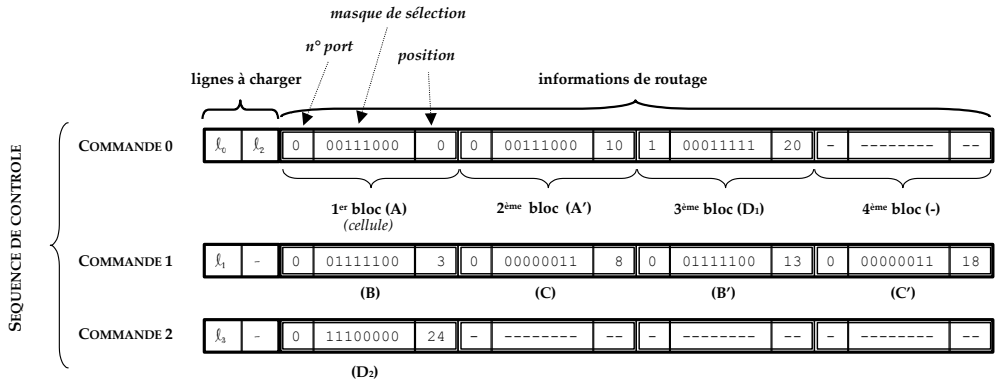


FIGURE 5 : EXEMPLE DE SEQUENCE DE CHARGEMENT

Une fois construite (ce qui nécessite plusieurs opérations, mais ce n'est pas un problème puisqu'elles peuvent être menées hors du chemin critique), la séquence de contrôle est enregistrée dans la table des séquences. Dans cet article, nous considérons une table à accès direct. Comme le prédicteur de branchements (dans nos modèles, nous avons prévu un prédicteur de type *gshare* [1]), cette table est indexée par un XOR du PC courant et de l'historique global des branchements, ce qui permet d'exploiter une associativité de chemins [2]. Afin d'éviter les problèmes d'alias et atteindre une qualité de prédiction suffisante, la table de séquences est étiquetée avec le PC.

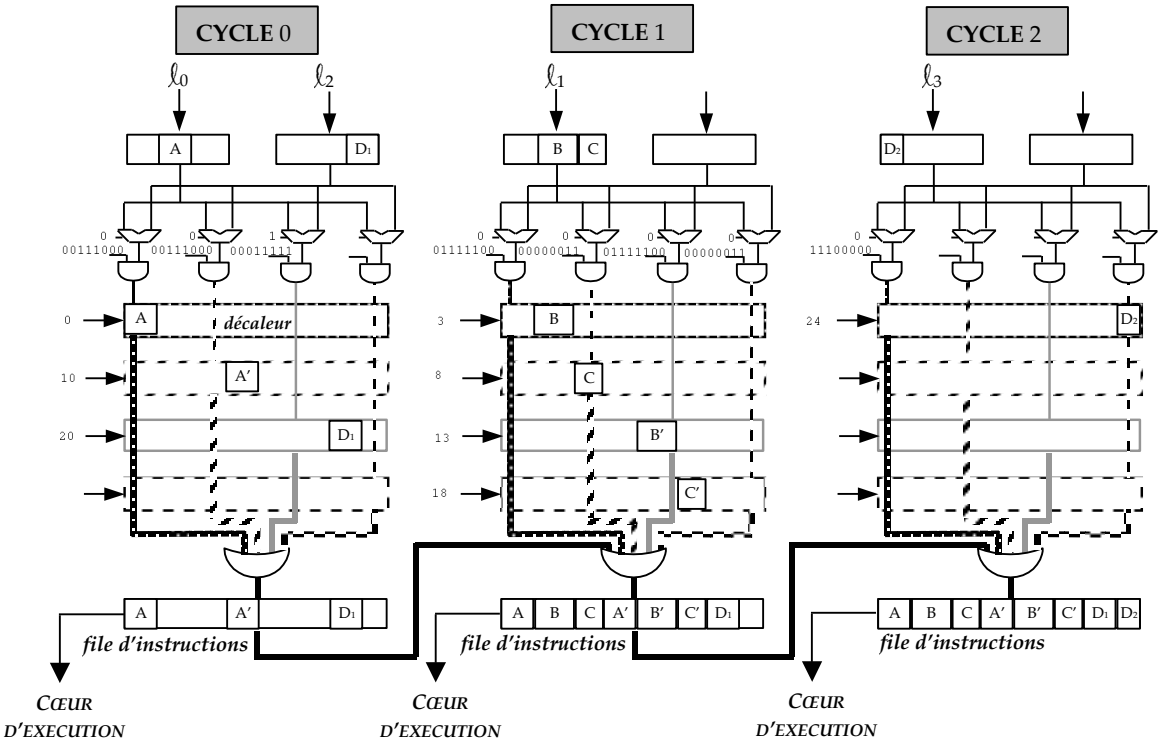


FIGURE 6 : LOGIQUE DE REALIGNEMENT

4. Résultats

Nous avons évalué les performances de notre mécanisme par simulation, à l'aide de l'outil *sim-fast* de la chaîne SimpleScalar (référence). Le processeur modélisé dispose d'un cœur d'exécution idéal, capable d'absorber le débit d'instructions délivré par le frontal. Ses paramètres sont :

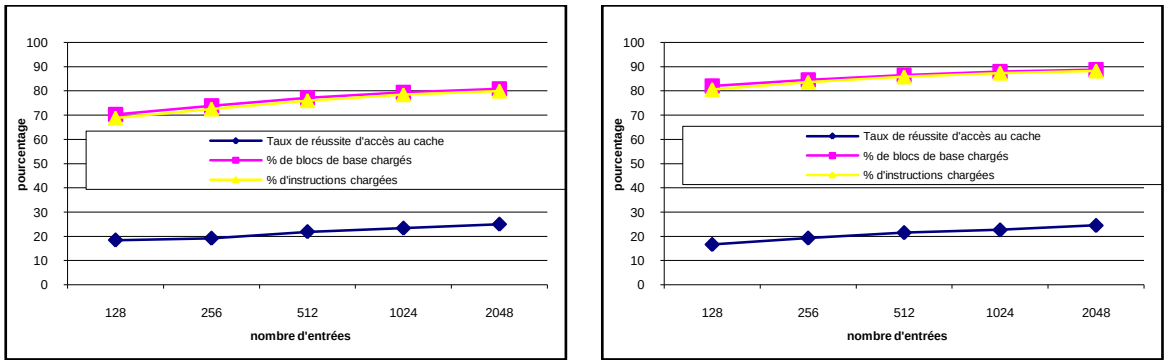
Cache d'instructions	8K-lignes adressées directement / 8 instructions par ligne / 4 ports / 8 bancs / pénalité de défaut de 10 cycles
Prédicteur de branchement ▪ *prédiction de la direction (PHT)* ▪ *prédiction de l'adresse cible (BTB)*	type *gshare* / table de 8K-entrées parfaite
Cœur d'exécution	idéal, pénalité de mauvaise prédiction de branch[t] de 7 cycles
Unité de construction des séquences de chargement	pipelinée, 3 étages

4.1 Taux de couverture

Tout d'abord, nous avons mesuré le *taux de succès* des accès à la table de séquences de contrôle et le *taux de couverture* de notre mécanisme, c'est-à-dire les pourcentages d'instructions et de blocs de base chargés depuis le cache par l'intermédiaire d'une séquence de contrôle. Ces résultats, en fonction de la taille de la table de séquences, sont représentés sur la figure 7, pour des fenêtres de chargement (longueur des séquences) de 2 ou 4 cycles. Bien que le taux de succès des accès à la table soit faible (inférieur à 25%), le taux de couverture des instructions est conséquent : de 65% à 80% pour une fenêtre de 2 cycles à plus de 80% pour une fenêtre de 4 cycles. Ceci est dû au fait qu'un seul succès permet de gérer les accès au cache pendant plusieurs cycles. On note que le taux de couverture des blocs de base est très proche de celui des instructions, ce qui semble indiquer que notre mécanisme ne sert pas une catégorie particulière de blocs (par exemple, les plus courts ou les plus longs).

4.2 Performances de la prédiction de branchement

Comme nous l'avons expliqué au paragraphe 3.2, notre mécanisme (prédiction *1 bit*, avec table étiquetée, donc sans problème d'alias), associé au prédicteur de branchement classique (*gshare*), forme un prédicteur hybride. La table 1 montre les performances respectives des deux composantes de ce prédicteur hybride (évaluées de manière individuelle) : elles indiquent qu'un prédicteur 1-bit sans collision est souvent meilleur que le prédicteur *gshare*.



Fenêtre de 2 cycles ***Fenêtre de 4 cycles***

FIGURE 7 : TAUX DE SUCCES D'ACCES A LA TABLE ET TAUX DE COUVERTURE DES INSTRUCTIONS

	bzip2	*vortex*	*gcc*	*parser*	*twolf*	*vpr*	*gzip*	*eon*	*MOY.*
***gshare*, table de 8 K-entrées**	1,15	2,73	8,72	5,93	8,45	13,27	6,85	1,31	**6,15**
prédiction 1 bit, table 8 K-entrées étiquetée	1,75	0,94	4,43	5,7	6,72	12,83	8,19	2,15	**5,29**

TABLE 1 : TAUX D'ERREURS D'UN PREDICTEUR 1-BIT ETIQUETE ET D'UN PREDICTEUR *GSHARE*

Par ailleurs, l'utilisation de séquences de contrôle fait l'hypothèse que le chemin suivi sera le même que la fois précédente, ce qui équivaut à une prédiction multiple 1 bit sans collision. La table 1 laisse espérer des résultats satisfaisants. La table 2 présente les taux d'erreur de prédiction du prédicteur *gshare* (table de 8K-entrées) et de notre mécanisme (table de 512 entrées), pour 3 largeurs de fenêtres de chargement. On observe que, parfois, les prédictions de *ghsare* sont moins fiables que celles issues des séquences de contrôle, et ce malgré l'utilisation d'un historique de branchement plus long (13 bits contre 9). Ceci est principalement dû au fait que, la table des séquences étant étiquetée, notre mécanisme ne prédit que les branchements fortement corrélés à l'historique.

4.4 Débit de chargement des instructions

Nous avons comparé notre mécanisme (table de séquences de 512 entrées, fenêtre de chargement de 3 cycles) avec un frontal classique comportant : a) un prédicteur *gshare* standard, avec une table de 8K-entrées, b) un prédicteur multiple simulé par un *gshare* à 8K-entrées, appelé 3 fois par cycle, c) le même prédicteur multiple associé à la technique de combinaison des accès redondants au cache (les prédictions étant réalisées « à la volée » à chaque cycle, il n'y a pas de réordonnancement possible). La figure 8 montre le nombre moyen d'instructions chargées par cycle. En moyenne, un prédicteur *gshare* permet de lire 5 instructions par cycle, alors qu'un prédicteur multiple n'atteint que 8 instructions par cycle à cause des conflits de bancs dans le cache. Ajouter à ce prédicteur la possibilité de combiner des accès redondants permet d'obtenir un débit de chargement de 19 instructions par cycle, en moyenne (mais n'oublions pas que, sur le chemin critique, un tel prédicteur sollicité trois fois en un cycle n'est pas réalisable). La combinaison mise en œuvre dans notre mécanisme conduit au même débit, et l'intégration du réordonnancement permet de dépasser 30 instructions par cycle.

	fenêtre de charg^t = 2 cycles			fenêtre de charg^t = 3 cycles			fenêtre de charg^t = 4 cycles		
	gshare	séquences	moyenne	*gshare*	séquences	moyenne	*gshare*	séquences	moyenne
gcc	12.80	5.74	7.56	13.36	5.40	7.24	13.15	7.60	9.00
parser	7.06	8.71	8.65	8.08	7.94	7.94	8.53	10.20	10.13
vortex	6.02	4.17	4.46	6.43	3.29	3.71	6.66	6.19	6.25
vpr	13.61	13.89	13.86	13.83	13.64	13.66	13.08	18.12	17.39
twolf	11.49	8.54	9.03	12.48	7.94	8.61	11.90	11.29	11.39
eon	1.57	5.73	5.22	1.40	4.91	4.60	1.25	4.17	3.92
bzip2	4.36	1.71	1.73	9.39	1.93	1.95	4.34	1.68	1.70
gzip	7.85	8.88	8.86	8.14	8.88	8.87	8.08	9.56	9.52
MOYENNE	**9.50**	**7.37**	**7.62**	**10.38**	**6.85**	**7.20**	**10.28**	**8.74**	**8.92**

TABLE 2 : TAUX DE MAUVAISES PREDICTIONS DE BRANCHEMENTS

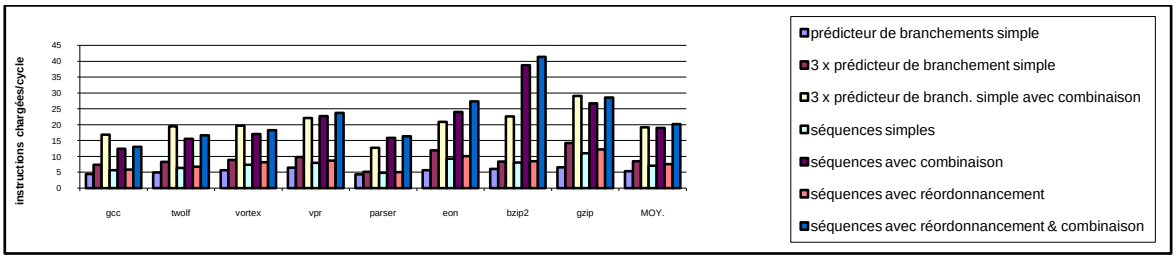


FIGURE 8 : DEBIT DE CHARGEMENT DES INSTRUCTIONS

5. Conclusion

Nous avons proposé une optimisation des accès au cache permettant d'accroître très sensiblement le débit de chargement des instructions sans allonger le chemin critique. Elle consiste à combiner plusieurs accès à une même ligne de cache et à réordonner les accès sur plusieurs cycles afin de réduire le nombre de conflits de bancs. Nous avons montré qu'avec un cache parfait et des accès optimisés sur 3 cycles, les performances étaient améliorées d'un facteur 6. Avec un cache plus réaliste, le débit de chargement obtenu avec une table de séquences de chargement de 512 entrées est d'environ 20 instructions par cycle, ce qui devrait satisfaire quelques futures générations de processeurs. Ce mécanisme pourrait encore être amélioré en choisissant une meilleure heuristique de remplacement pour éviter la redondance des séquences. Enfin, un cœur d'exécution capable de renommer les instructions dans le désordre exploiterait plus largement le débit fourni par notre mécanisme.

Références

[1] MCFARLING S., Combining Branch Predictors, *Tech. Report DEC-WRL TN-36*, 1993.

[2] MENEZES K., SATHAYE S., CONTE T., Path Prediction for High Issue Rate Processors, *International Conference on Parallel Architectures and Compilation Techniques*, 1997.

[3] RAKVIC R., BLACK B., SHEN J.P., Completion Time Multiple Branch Prediction for Enhancing Trace Cache Performance, *27th International Symposium on Computer Architecture*, 2000.

[4] ROTENBERG E., BENNETT S., SMITH J., Trace Cache: a Low Latency Approach to High Bandwidth Instruction Fetching, *29th International Symposium on Microarchitecture*, 1996.

[5] SEZNEC A., JOURDAN S., SAINRAT P., MICHAUD P., Multiple-block Ahead Branch Predictors, *7th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[6] YEH T.-Y., MARR D., PATT Y., Increasing the Instruction Fetch Rate via Multiple Branch Prediction and a Branch Address Cache, *International Conference on Supercomputing*, 1993.