Travaux Pratiques Archi SEOC-3A - 2021/2022
Sujet de Frédéric Pétrot et Arthur Perais

Implantations et évaluations de quelques prédicteurs de branchements

Durée : 4 heures

1 Organisation

Ce TP se fait soit seul, soit en binôme, à votre convenance. Le travail demandé TP fait l'objet d'un rendu contenant un court rapport qui inclut les *résultats* d'expérimentations et un *tarball* (ou un git) contenant vos sources à fournir en fin de séance. Un gabarit en LaTeX qui reprend le predicteur BHT avec un compteur à saturation est donné en exemple. Vous avez également un script fourni pour vous permettre de systématiser les simulations et l'obtention des graphes des résultats. Ce script marche avec le prédicteur BHT qui a deux arguments, il est donc nécessaire de le modifier pour utiliser un prédicteur avec un nombre d'arguments différents. Le sous-répetoire `sle3a/rendu` contient un exemple de .tex permettant de faire un petit rapport sur le travail. **Vous êtes invités à lire tous le sujet avant de commencer le TP.**

2 Introduction

On cherche à étudier le comportement en situation réelle de différents prédicteurs de branchement, du plus simple au plus complexe. Pour cela, on va développer un modèle en C++ de chacun de ces prédicteurs sur lequel on fera passer un ensemble de traces d'exécutions, nous permettant d'obtenir en sortie le ratio de mauvaises prédictions (en fait le nombre de *Miss Prediction per Kilo Instructions*, MPKI). On fera varier les différents paramètres du prédicteur, en particulier la taille des tables, afin d'en extraire un comportement asymptotique (par benchmark), dont on tirera un graphe.

L'infrastructure dans laquelle insérer le modèle de prédicteur est celle qui a été utilisée par le *The 5th JILP Championship Branch Prediction Competition (CBP-5)*[1]. Cette infrastructure effectue la lecture des traces et l'appel à des fonctions qui implantent le prédicteur, et donne en sortie diverses statistiques dont les MPKI.

Le prédicteur par défaut dans l'archive fournie est un prédicteur bimodal à saturation « n-bits » vu en cours et rappelé en Figure 1 pour le cas 2-bit. L'automate régissant chaque compteur est rappelé en Figure 2.

Dans une case donnée, un branchement « prédit pris » aura une valeur $v \geq 2^{n-1}$ et « prédit non pris » aura une valeur $v < 2^{n-1}$ (on note qu'il s'agit d'une convention, on pourrait très bien faire l'inverse). Nous ferons en séance une analyse rapide du code afin de comprendre comment il a été implanté dans l'infrastructure.

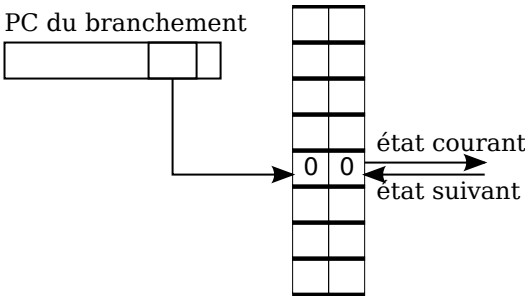


Figure 1 – Prédicteur bimodal

3 Travail demandé

On implantera différents prédicteurs en prenant soin de paramétrer les tailles afin de pouvoir lancer facilement plusieurs exécutions (*cf.* l'exemple fourni) et ainsi pouvoir tracer des courbes et voir les asymptotes. **Important :** Les figures montrent souvent des prédicteurs de petites tailles, mais vous êtes invités à considérer des prédicteurs commençant au

1. Dont on trouvera la version originale sur `https://www.jilp.org/cbp2016/framework.html`

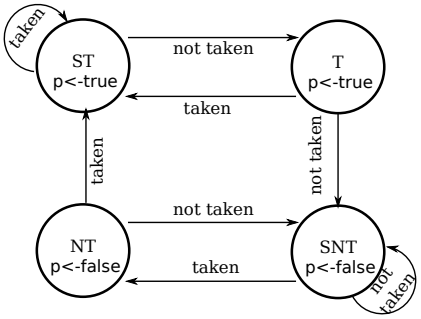


FIGURE 2 – Automate d'un compteur 2-bit bimodal

moins à 128 entrées (log2(128) = 7). Lors de l'évaluation de chaque prédicteur, vous êtes invités à analyser son comportement en isolation (e.g., un prédicteur de plus grande taille est-il toujours meilleur ?) mais aussi à le comparer aux autres types de prédicteurs (e.g., deux prédicteurs aux performances égales mais l'un requiert beaucoup plus d'entrées que l'autre).

3.1 Prédicteurs simples

On commencera par les prédicteurs relativement simples vus en cours :

1. un prédicteur *gshare* qui utilise un historique global sur H bits qui est combiné à l'adresse du branchement à prédire via un XOR, comme illustré sur la Figure 3.

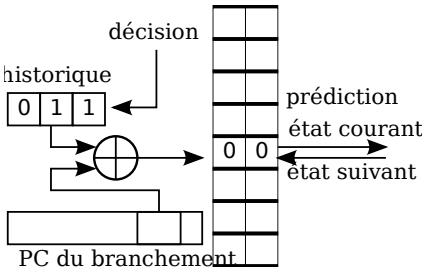


FIGURE 3 – Prédicteur gshare

2. un prédicteur local qui utilise une table d'historiques sur H bits (la *Pattern History Table*) indexée par les poids faibles de l'adresse du branchement. En utilisant l'entrée d'historique, on indice une BHT bimodale avec compteurs 2-bit de taille 2^H, comme illustré sur la figure 4 ;

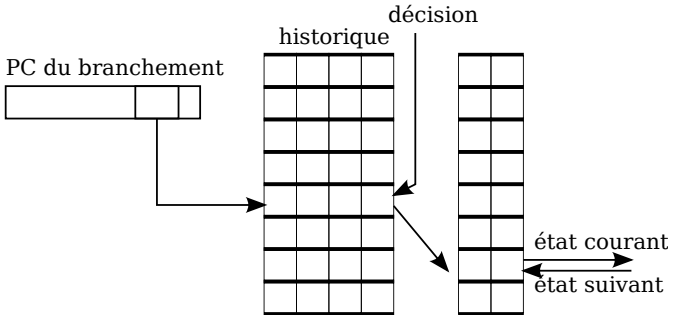


FIGURE 4 – Prédicteur local

3. une combinaison de deux prédicteurs et le métaprédicteur associé. On va ici combiner un prédicteur global *gshare* avec un historique de $H_g = 10$ (soit 1024 entrées), et un prédicteur *2-level* local comme celui de la précédente question, avec $H_l = 10$ par entrée de la PHT (soit 1024 entrées pour la BHT, 1024 entrées pour la PHT). La BHT indexée est un simple prédicteur bimodal.

Le choix du prédicteur à utiliser se fait grâce au métaprédicteur qui, comme les BHT des "vrais" prédicteurs, contient 1024 entrées de compteurs 2 bits. On accède au compteur en utilisant l'adresse du branchement à prédire, et ce compteur nous dit simplement quel prédicteur du prédicteur global ou du prédicteur local va effectuer la prédiction. Le compteur est incrémenté lorsque le prédicteur qui a prédit est correct et l'autre prédicteur a fait le mauvais choix, et est décrémenté dans le cas opposé (*cf.* la machine d'état de la figure 5). Il n'est pas nécessaire de rendre le prédicteur paramétrable. *Ici, on ajoutera des statistiques afin de compter les bonnes et mauvaises prédictions de chaque prédicteur respectif, ainsi que le nombre de prédictions effectivement utilisées (i.e., sélectionnées par le métaprédicteur) de chaque prédicteur.*

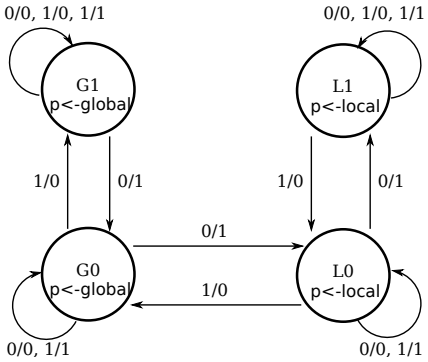


FIGURE 5 – Machine à états du choix du prédicteur. Légende : $global/local$, 0 décision incorrecte, 1 décision correcte

3.2 Perceptron

On se propose ensuite d'implémenter un prédicteur moderne tel que le prédicteur neural à base de perceptrons de Jiménez et Lin [2], et en comparer la performance avec les prédicteurs de la première partie.

Ce prédicteur utilise la version la plus simple des réseaux de neurones appelés perceptrons, puisqu'il n'utilise qu'une seule couche de neurones. Un perceptron "apprend" une fonction booléenne à n entrée, et dans le cas du prédicteur de Jiménez et Lin, ces n entrées correspondent aux bits de l'historique global de branchement (le même que dans *gshare*). La sortie étant la direction prédite pour le branchement. Le modèle mathématique utilisé pour un tel perceptron est donné dans la Figure 6. Dans la Figure, y dénote un vecteur de poids w_i (entiers signés), et le calcul effectué par le perceptron est le produit scalaire des entrées avec le vecteur de poids. Dans ce modèle, on ajoute une entrée x_0 valant toujours 1 pour biaiser le résultat. Dans ce modèle, la valeur des entrées x_i est soit -1 (branchement non pris) soit 1 (branchement pris). Un résultat de la somme négatif est interprété comme la prédiction "non-pris", et un résultat positif est interprété comme la prédiction "pris".

$$y = w_0 + \sum_{i=1}^{n} x_i w_i.$$

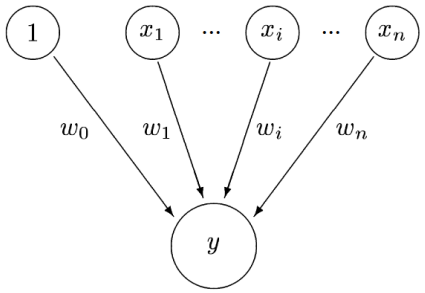


FIGURE 6 – Perceptron

2. Lien vers "Dynamic Branch Prediction with Perceptrons".

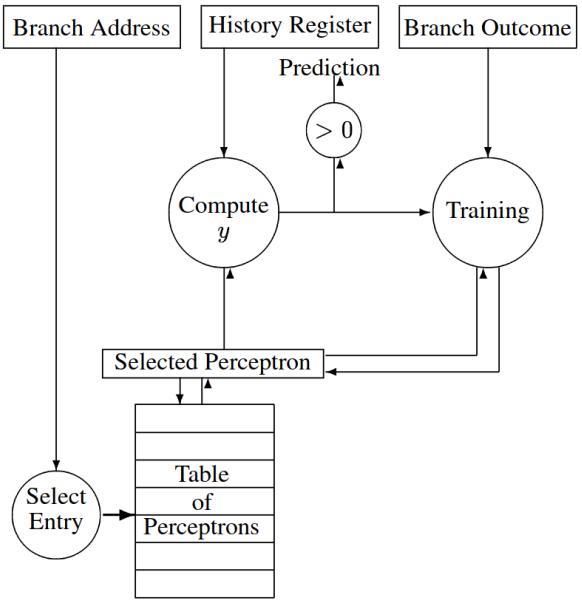


FIGURE 7 – Prédicteur neural à base de perceptrons

Le diagramme du prédicteur, emprunté au papier de Jiménez et Lin est donné en Figure 7. Afin de prédire la direction d'un branchement conditionnel, un tableau de perceptrons est indexé via l'adresse du branchement à prédire. Chaque entrée de la structure modélise un perceptron tel que décrit dans le paragraphe précédent. Pour générer la prédiction, le prédicteur va donc effectuer le produit scalaire du vecteur de poids (le perceptron) et le contenu du GHR (sans oublier le biais !)

Une fois la direction réelle du branchement obtenu, il convient de mettre à jour le prédicteurs. Ici, cela revient à mettre à jours les poids du vecteur de poids du perceptron utilisé pour prédire. On utilisera l'algorithme suivant :

Soit $t = -1$ si le branchement est non-pris, 1 sinon

Soit θ le *seuil*, un paramètre de l'algorithme de mise à jour

Soit $y_{out} = \begin{cases} 1, & \text{si } y > \theta \\ 0, & \text{si } -\theta \leq y \leq \theta \\ -1, & \text{si } y < \theta \end{cases}$

si $y_{out} \neq t$ alors :
  pour $i := 0$ à n, faire :
    $w_i := w_i + t.x_i$
  finpour
finsi

L'article de Jiménez et Lin suggère d'utiliser un historique global de branchement entre 12 et 62 bits suivant le budget matériel (en KB), des poids de 7 à 9 bits, mais ne donne pas de valeur pour θ.

Dans le cadre de ce TP, on se limitera à un historique global de même taille que pour gshare, avec un nombre de perceptrons similaires, et on s'attachera à comparer la performance des deux prédicteurs, car il s'agit de deux prédicteurs à corrélation globale. Avant d'effectuer cette analyse, il conviendra cependant de trouver un θ donnant de bonnes performances, et éventuellement de donner une intuition sur pourquoi.

3.3 Aller plus loin

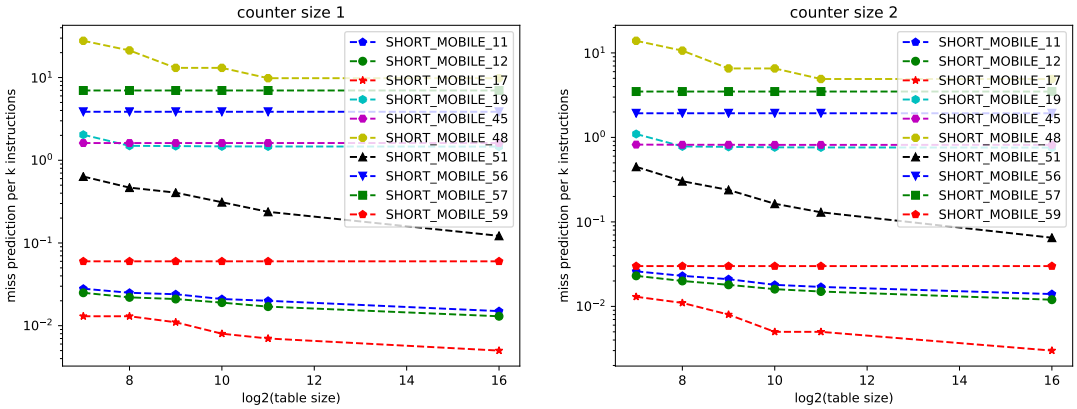
L'idée de cette dernière partie est de choisir un des prédicteurs implémentés dans les parties précédentes et de chercher à minimiser les MPKI par tous les moyens possibles : taille des tables, fonction de hachage, associativité, combinaisons de prédicteurs, type d'historique (dans la mesure de ce que permet le simulateur), tout est permis. Ici, on vous encourage à analyser les traces, par exemple en construisant des statistiques sur chaque branchement dans la fonction *UpdatePredictor()*, afin de tenter de déterminer pourquoi un branchement est mal prédit alors qu'il a l'air régulier ou corrélé à un voisin. Dans votre rapport, il conviendra d'expliquer les techniques utilisées et vos analyses quand à ce qui fonctionne le mieux.

4 Infrastructure

Il y a un script `sle3a-doit.sh` dans le répertoire `script` qui lance automatiquement l'exécution sur un sous-ensemble des traces pour un prédicteur donné, qu'il a fallut préalablement compiler dans le répertoire `sim`. Il y a dans le script pour le prédicteur n-modal 2 boucles imbriquées : la boucle externe, indice i, fait varier le nombre de bits du compteur, et la boucle interne, d'indice j, fait varier le nombre de bits de PC à utiliser pour indexer les tables. Vous serez amené à modifier les valeurs des bornes en fonction des prédicteurs, voir à changer un peu cette partie si vous faites des prédicteurs exotiques.

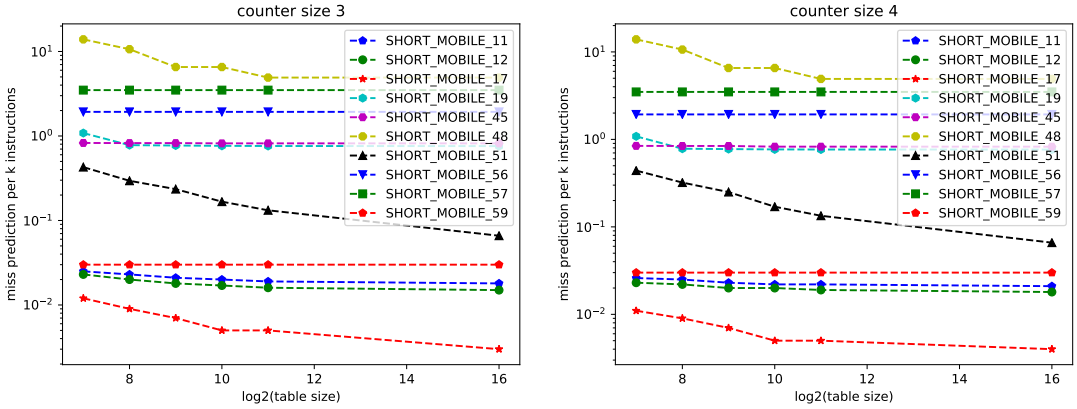
Le script `sle3a-doit.sh` produit ses résultats dans `../results/xxx`, ou `xxx` est un répertoire dont le nom peut-être choisi à l'envie. Il les analyse ensuite et génère des programmes `python` qui servent à tracer des courbes (avec `matplotlib`). Ces courbes sont crées dans le répertoire où on appelle `sle3a-doit.sh` et ce sont ces courbes que l'on vous demande d'analyser.

On donne sur la figure 8 le type de résultat attendu pour le prédicteur n-bit.



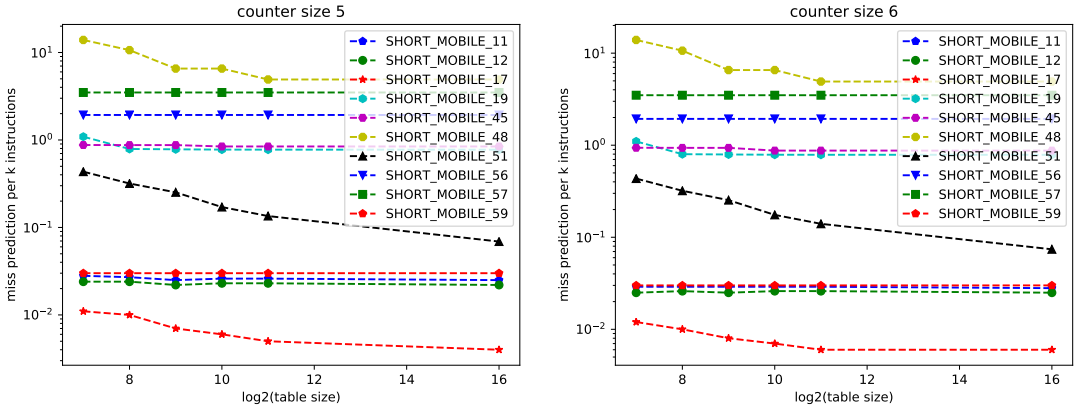
(a) 1 bit par compteur

(b) 2 bits par compteur



(c) 3 bits par compteur

(d) 4 bits par compteur



(e) 5 bits par compteur

(f) 6 bits par compteur

FIGURE 8 – Les courbes de MPKI pour le prédicteur bimodal simple, en fonction du nombre de bits par compteur et du nombre d'entrées.