Analysis of a tag-based branch predictor

Pierre Michaud

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Analysis of a tag-based branch predictor

Pierre Michaud

N°5366

Novembre 2004

Systèmes communicants

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--5366--FR+ENG

Analysis of a tag-based branch predictor

Pierre Michaud

Systèmes communicants
Projet CAPS

Rapport de recherche n° 5366 — Novembre 2004 — 21 pages

Abstract: The method most often used for inventing new branch predictors is to start from a known predictor and try to improve it. However, this method gives little insight in the frequent case where we fail to improve the predictor. This study proposes a new approach, which we think provides a better understanding. We start from a model of ideal predictor, and introduce successive degradations, until we obtain a predictor that can be implemented in hardware. On each degradation, it is possible to quantify the loss, analyze the reasons for it, and sometimes propose remedies. This paper is an illustration of this method on the family of tag-based predictors derived from PPM.

Key-words: branch prediction analysis, PPM, limited-size predictor, sequence frequency, up-down saturating counter

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Analyse d'un prédicteur de branchements

Résumé : La méthode la plus souvent utilisée pour inventer de nouveaux prédicteurs de branchements est de partir d'un prédicteur connu et d'essayer de l'améliorer. Cependant, lorsque cette méthode échoue, elle se prête difficilement à une analyse permettant de comprendre les raisons de l'échec. Nous proposons une nouvelle méthode, qui n'a pas cet inconvénient. Nous partons d'un modèle de prédicteur idéal, et nous introduisons des dégradations successives correspondant aux contraintes matérielles, jusqu'à obtenir un prédicteur réaliste. Chaque dégradation peut être quantifiée, analysée, et parfois, corrigée. Cette étude est une illustration de cette méthode sur la famille des prédicteurs dérivés du prédicteur idéal PPM.

Mots-clé : analyse de prédiction de branchements, PPM, prédicteur de taille limitée, fréquence de séquence, compteur bidirectionnel saturant

1 Introduction

Branch mispredicts are an important source of performance loss in modern general purpose processors. The goal of branch prediction research is to propose new branch predictors that have a lower mispredict rate, or that are simpler to implement, or both. The limits of branch prediction (under certain hypotheses) have been studied in [2, 5], but these previous limit studies did not consider predictor storage limitations, which is an important parameter. And indeed, recent branch prediction research shows that, given a limited storage, there is still room for improvement [14, 8, 10, 13], although this often requires more complex predictors. Yet, it is not clear how small the mispredict rate can be for a fixed hardware complexity. Answering this question is difficult. Branch prediction is not easy to theorize because applications are difficult to model. Moreover, hardware complexity cannot be quantified easily with a single number. These are probably some of the reasons why inventing new branch predictors is more an art than a science.

The method most often used can be described as follows. Typically, one has an intuition about a possible way to decrease the mispredict rate of a known predictor. Then the idea is tested by running simulations, and the modified predictor is compared with the reference predictor. If the mispredict rate is not significantly improved (which is a frequent situation), there are several possibilities, among which :

- The intuition was wrong : understanding why it was wrong should be instructive.
- The intuition was not completely wrong, but the idea must be combined with another (yet unknown) idea for its potential to be exposed
- The reference predictor cannot be improved

It is difficult to distinguish between these possibilities, precisely because the mispredict rate does not change. Yet, we believe it is important to know the asnwer. This paper proposes a method that does not have this drawback. We start from a model of ideal predictor based on a few basic principles. Then we introduce successive degradations until we obtain a predictor that can be implemented in hardware. When the impact of a degradation is more pronounced than expected, we can go back, analyze what happened, and try to bring a correction. This study is an illustration of the method on the family of tag-based predictors derived from PPM [2].

In Section 3, we describe an ideal predictor that can serve as a model for a large class of tag-based predictors. Section 4 introduces successive degradations which are analyzed systematically. The predictor finally obtained is then compared with *2bcgskew*.

2 Definitions and notations

We define the program control flow (PCF) as a finite sequence $(B_i)_{i=1}^{N}$ of dynamic basic blocks $B_i \in \mathcal{B}$ where $\mathcal{B}$ is the set of all static basic blocks constituting the program text. Typically, the PCF corresponds to the whole program execution, or to the execution of a

program between consecutive context switches. The PCF length N is typically large, e.g., several hundreds of thousands of dynamic basic blocks. We assume that the prediction of block B_{j+1} is based only on the knowledge of blocks B_1 to B_j. One may use information other than just the PCF, for example instructions dependencies [17]. However, most predictors proposed so far are PCF-based predictors, and we focus on these.

We denote $S(n)$ the set of all possible sequences of n consecutive blocks. In particular, $S(1) = \mathcal{B}$. Given $m < n$ and two sequences $u \in S(m)$ and $s \in S(n)$, notation $u \prec s$ means that sequence s ends on sequence u. For each sequence s, we define its *frequency* $f(s)$ as

$$f(s) = \frac{\text{number of occurrences of } s \text{ in the PCF}}{N}$$

which implies $f(s) \leq 1$. For simplifying the study, we assume that each block $B \in \mathcal{B}$ has only two possible successor blocks in the PCF, which is often the case in practice. Hence there exists at most $|S(n)| \leq 2^n$ possible sequences of length n. Throughout this study, 1_s and 0_s denote respectively the most frequent and least frequent successor block of sequence s in the PCF. In particular, for a block B, 1_B and 0_B are respectively its most frequent and least frequent successors. By definition, $\forall s \in S(n)$, we have $f(s.1_s) \geq f(s.0_s)$, where we denote $s.1_s$ (resp. $s.0_s$) the sequence from $S(n+1)$ formed by appending block 1_s (resp. 0_s) to sequence s. More generally, given two sequences $u \in S(m)$ and $s \in S(n)$, we denote $u.s$ the sequence from $S(m+n)$ formed by appending s to u.

We define the *stream* of a sequence s as the sequence of successor blocks of s in the PCF. For example, if sequence s is encountered four times in the PCF and these four occurrences are followed respectively by blocks 1_s, 1_s, 0_s and 1_s, the stream of s is 1101.

3 Model of ideal predictor

Our ideal predictor consists of a set T of sequences. The number of sequences in T is denoted $|T|$. Sequences in T may have a fixed length, or may have various lengths. However we assume that sequences lengths are strictly greater than one, i.e., $T \subseteq \bigcup_{n>1} S(n)$. We also assume that the content of T is fixed for a given PCF.

For each B_j in the PCF, we consider the sequences $B_{j-n+1} \cdots B_j$ of length n for all $n \in [2, j]$. If some of these sequences are in T, we define s as the longest of these matching sequences (one will recognize the PPM ordering of sequences [3, 2]). Otherwise, if there is no matching sequence, $s = B_j$. We predict B_{j+1} to be 1_s. Hence, if $B_{j+1} = 0_s$, this is a mispredict. We define the *mispredict frequency* $m(T) \leq 1$ as the total number of mispredicts divided by the PCF length N. For $n \geq 1$, we will denote $m(n)$ the quantity

$$m(n) = \sum_{s \in S(n)} f(s.0_s) \tag{1}$$

In particular $m(S(n)) = m(n)$ for $n > 1$, and $m(\varnothing) = m(1)$. It can be shown that, for $i \geq 0$,

$$m(n+i) \leq m(n) \tag{2}$$

i.e, $m(n)$ is non-increasing with n. This comes from the fact that a sum of mins does no exceed the min of sums :

$$\begin{aligned} m(n+i) &= \sum_{s\in S(n)} \sum_{u\in S(i)} \min(f(u.s.1_s), f(u.s.0_s)) \\ &\leq \sum_s \min(\sum_u f(u.s.1_s), \sum_u f(u.s.0_S)) \\ &= m(n) \end{aligned}$$

which prooves (2). This justifies PPM ordering. [1]

In the general case of a random T, and under PPM ordering, the mispredict frequency is

$$m(T) = m(1) - \sum_{s\in T}(f(s.0_u) - f(s.0_s)) \tag{3}$$

where u depends on s and is the longest sequence in $\mathcal{B} \cup T$ such that $u \prec s$. Expression (3) can be understood as follows. All the mispredicts are counted in the sum $m(1) + \sum_s f(s.0_s)$. We must remove from this sum the "false" mispredicts on sequences u, i.e., occurrences of u followed by 0_u but for which there exists a longer matching sequence in T. The sum $\sum_s f(s.0_u)$ corresponds to these false mispredicts.

3.1 Maximum sequence length

If T is finite, we have $m(T) \geq m(n)$, where n is the length of the longest sequence in T. This can be seen as follows. Let us add to T all the sequences from $S(n)$ that are not already in T. We obtain a set $R \supseteq T$. Referring to formula (3) applied to set R, we have

$$m(R) = m(T) - \sum_{s\in R-T} (f(s.0_u) - f(s.0_s))$$

which implies $m(R) \leq m(T)$. As the longest matching sequence in R is always of length n, i.e., $m(R) = m(n)$, we have $m(T) \geq m(n)$.

3.2 Removing useless sequences

For a limited $|T|$, we seek an optimal set T of sequences that minimizes $m(T)$.

Referring again to formula (3), let us assume that there exists an $s \in T$ such that $0_u = 0_s$. We have $f(s.0_s) = f(s.0_u)$. If we remove s from T, the only change on $m(T)$ may come from

[1] It should be noted that selecting the longest matching sequence is not always the best choice. For example, let us consider two sequences $u \in T$ and $s \in T$ such that $u \prec s$. Let us consider the susbtream X of s corresponding to occurrences of s as the longest matching sequence. As X is not the whole stream of s but only a substream, the most frequent block in X may be 0_s instead of 1_s. If this is the case, and if $0_s = 1_u$, it is better to select u instead of s. In other words, adding a sequence to T may increase $m(T)$.

sequences v in T for which s is the longest matching subsequence. However, after removing s, the longest matching subsequence of sequence v becomes u. As $0_u = 0_s$, the contribution of v does not change. Hence removing s keeps $m(T)$ unchanged.

We denote $T^* \subseteq T$ the set of sequences remaining after removing from T all the sequences s such that $0_u = 0_s$, where u is the longest sequence in $\mathcal{B} \cup T$ such that $u \prec s$. Expression (3) becomes

$$\begin{aligned} m(T) &= m(T^*) \\ &= m(1) - \sum_{s \in T^*} (f(s.1_s) - f(s.0_s)) \end{aligned} \tag{4}$$

It should be noted that $T^{**} = T^*$, which means that for all $s \in T^*$ and u the longest sequence in $\mathcal{B} \cup T^*$ such that $u \prec s$, we have $1_s = 0_u$.

3.3 Single sequence length

Let us first consider the case which constrains sequences in T to have a fixed length n. Expression (3) becomes

$$m(T) = m(1) - \sum_{s \in T} (f(s.0_B) - f(s.0_s))$$

where B is the last block in s. For a limited $|T|$, one can minimize $m(T)$ by sorting sequences in $S(n)$ in decreasing values of $f(s.0_B) - f(s.0_s)$, and putting in T the first $|T|$ sequences. As $|T|$ increases, $m(T)$ decreases from $m(1)$ to $m(n)$. Notice that $m(T)$ is necessarily a convex function of $|T|$ in this case.

3.4 Multiple sequence lengths

The general case which considers multiple sequence lengths in T is a more difficult optimization problem. At this point in the analysis, it is convenient to introduce a hypothesis, which will lead to a simple heuristic. We introduce the *convexity hypothesis*, which we define as follows :

$$\forall \quad u, s : u \prec s, \quad f(u.1_u) - f(u.0_u) \geq f(s.1_s) - f(s.0_s)$$

Let S be the set of sequences that are allowed to be in T. It can be shown that, in order to minimize $m(T)$, we only need to consider sequences in S^* ($S^* \subseteq S$ is obtained as previously, by removing from S all sequences $s \in S$ such that $1_s = 1_u$, where u is the longest sequence in $\mathcal{B} \cup S$ such that $u \prec s$). This can be seen as follows. Let us consider a $T^* \subseteq S$ that is not included in S^*, and $s \in T^*$ such that $s \notin S^*$. Let u be the longest sequence in $\mathcal{B} \cup S^*$ such that $u \prec s$. Because $s \notin S^*$, we must have $1_s = 1_u$. Then, because $s \in T^{**}$, u cannot be in $\mathcal{B} \cup T^*$. Let us define $R = T^* \cup \{u\}$. We have $s \notin R^*$, i.e., R^* is obtained from T^* by replacing s with u. From the convexity hypothesis and from (4), we must have

$m(R^*) \leq m(T^*)$. In summary, we can replace sequences not in S^* by sequences that are in S^*, yet without increasing $m(T^*)$.

We are now searching an optimal $T \subset S^*$. Under the convexity hypothesis, one can minimize $m(T)$ for a limited $|T|$ by sorting sequences $s \in S^*$ in decreasing values of $f(s.1_s) - f(s.0_s)$ and putting in T the first $|T|$ sequences. Notice that $T = T^*$ is a consequence of the convexity hypothesis. On real worloads, we may observe $|T^*| < |T|$. However, we should observe $|T^*| \approx |T|$ as an empirical validation of the convexity hypothesis, i.e., function $m(T)$ of $|T|$ should be approximately convex.

4 Degrading the ideal predictor

This study focuses on conditional branches, which constitute the majority of mispredicts, i.e., we consider only the sequences which last block ends on a conditional branch. Block 1_s corresponds to the most frequent branch direction, either *taken* or *not-taken*. Block 0_s corresponds to the other direction.

A first degradation we introduce in the ideal predictor is in the way sequences are represented. We represent a sequence with the address of the conditional branch ending the last block and with the global history of conditional branch directions, i.e., a bit vector representing the directions of the last $n-1$ conditional branches (the case $n = 1$ corresponds to the usual "bimodal" predictor [15, 11]). This is how sequences are represented in most predictors. There is some aliasing on sequences [12], yet it is easy to implement in hardware. This assumption also eases our simulations. We assume a maximum sequence length n_{max} and, for each static branch encountered in the PCF, we maintain a binary tree of depth n_{max} where each node represents a distinct global history value, which length corresponds to the depth of the node in the tree. Throughout this study, we assume $n_{max} = 41$.

Experimental set-up. We focused our attention on seven benchmarks. One is *go*, from the SPEC95 suite, and the six others are *gzip*, *vpr*, *gcc*, *crafty*, *parser*, and *twolf* from the SPEC2000 suite. We chose these benchmarks because of their mispredict rate, which is likely to be a performance bottleneck. We used the train input on all benchmarks. We skipped the first 100 millions instructions and collected conditional branches for the next 10 millions instructions (unless specified otherwise). To save space, we use *go* as our example PCF.

4.1 Limited number of sequence lengths

In the remaining, S still denotes the set of sequences allowed in T. We assume that $S = \bigcup_{n \in L} S(n)$, where $L \subseteq [2, n_{max}]$. Figure 1 shows $m(T)$ on *go* as a function of $|T|$ for $L = \{41 - 4 \times i\}_{i=0}^{9}$, for $L = \{6, 11, 21, 41\}$ for $L = \{11, 41\}$ and for $L = \{41\}$ (the curves show only values of $|T|$ corresponding to sequences such that $f(s.1_s) - f(s.0_s) \geq 4/N$). On all benchmarks, the curve of $m(T)$ in function of $|T|$ is approximately convex. However, we verified that they are locally not convex (except $L = \{41\}$ which is always convex). The convexity hypothesis seems to provide a good heuristic in practice. On the example

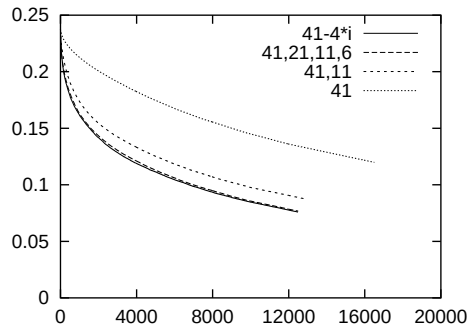


Figure 1: Example PCF : $m(T)$ as a function of $|T|$, for $L = \{41 - 4 \times i\}_{i=0}^{9}$, for $L = \{6, 11, 21, 41\}$ for $L = \{11, 41\}$ and for $L = \{41\}$.

PCF, four lengths $L = \{6, 11, 21, 41\}$ is sufficient. while a single length $L = \{41\}$ is clearly insufficient. Analyzing the contribution to $m(1) - m(T)$ of each length in $L = \{6, 11, 21, 41\}$, we found that length 41 contributes little on this particular PCF. On this PCF, a single length $L = \{21\}$ would be better than a single $L = \{41\}$. However, the contribution of length 41 was significant on certain benchmarks, especially *parser*. Some predictors, e.g., McFarling's bimodal/gshare [11] or YAGS [4], use a single global history length. The need for multi-length configurations has already been pointed out [16]. The optimized *2bcgskew* predictor [14, 1] is an example of dynamic global-history based predictor that uses multiple history lengths. In the remaining, we assume $L = \{6, 11, 21, 41\}$.

4.2 Up-down saturating counters

A real branch predictor has no *a priori* knowledge of 1_s for a sequence s. The real predictor discovers the stream of s progressively and makes predictions based on occurrences of s encountered so far. In this kind of situation, real predictors often use up-down saturating counter associated with each sequence [15, 9, 18]. When the branch ending the sequence is taken, the counter is incremented, otherwise it is decremented. For predicting the branch direction, we look at the counter value sign : if the counter value is positive, we predict *taken*, otherwise we predict *not-taken*. It can be shown that if the stream of s behaves like a Bernoulli trial, an up-down saturating counter with a small number of bits is asymptotically close to optimal. Assuming a probability $p \leq 0.5$ for the occurrence of 0_s after s, the behavior of a $2C$-state saturating counter can be modeled by the Markov chain depicted below :

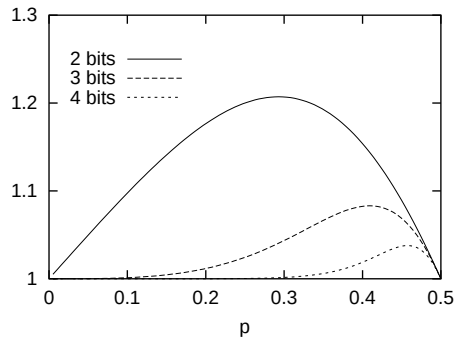
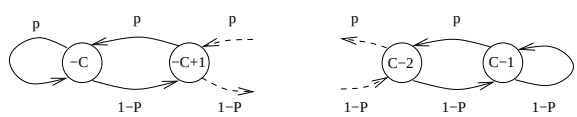


Figure 2: Up-down saturating counter : $M_C(p)/p$ as a function of $p \in [0, 0.5]$ for a 2-bit, 3-bit and 4-bit counter ($C = 2, 4, 8$).



The state is incremented with probability $1 - p$ and decremented with probability p. This corresponds to a one-dimensional random walk with reflecting barriers. A detailed analysis can be found in [6]. The Markov chain considered is ergodic, thanks to the saturating states. Hence, as the stream length increases, the probability to be in state k converges toward a limit value p_k independent of the initial state. It can be shown that probabilities p_k verify $p \cdot p_{k+1} = (1-p) \cdot p_k$. The probability P_0 that the state is strictly negative verifies equation $P_1 = 1 - P_0 = P_0 \cdot ((1-p)/p)^C$. The mispredict probability is $M_C(p) = (1-p) \cdot P_0 + p \cdot P_1$, i.e.,

$$M_C(p) = p + \frac{1-2p}{1 + (\frac{1-p}{p})^C} \tag{5}$$

Figure 2 shows the ratio $M_C(p)/p$ for $p \in [0, 0.5]$ for a 2-bit, 3-bit and 4-bit counter ($C = 2, 4, 8$). The 2-bit counter is close to optimal for $p < 0.05$, however its accuracy degrades for higher values of p. For instance, at $p = 0.3$, the 2-bit counter generates about 20% more mispredicts than the optimal predictor. If the stream is long enough, a 4-bit counter can be considered close to optimal.

To evaluate the impact of using saturating counters in practice, we ran two-pass simulations. During the first pass, we build the tree, compute frequencies, and sort sequences in S^* as previously. We compute the mispredict frequency during the second pass. Set T is still fixed for the whole PCF. We associate a saturating counter with each sequence in $\mathcal{B} \cup T$. The counter associated with a sequence is initialized according to the branch direction on

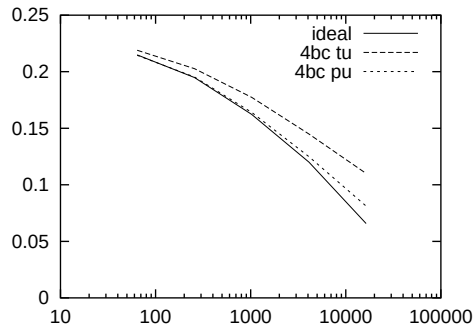


Figure 3: Example PCF : mispredict frequency in function of $|T|$ when using 4-bit counters with total update ("tu") and partial update ("pu") strategies, compared with the ideal predictor.

the first occurrence of the sequence in the PCF, i.e., state 0 if the branch is taken, state -1 otherwise. On the following occurrences of a sequence, the counter is incremented when the branch is taken, decremented otherwise. On the first occurrence of a static branch in the PCF, we predict a random direction. Otherwise, the prediction is given by the sign of the counter associated with the longest matching sequence in $\mathcal{B} \cup T$ that has been **encountered at least once** in the past. For updating the counters, we tried two strategies. The *total update* strategy updates the counters of all the matching sequences in $\mathcal{B} \cup T$. The *partial update* strategy updates only the counter of the longest matching sequence in $\mathcal{B} \cup T$.

Figure 3 shows the mispredict frequency in function of $|T|$ when using 4-bit counters with total update and partial update strategies, compared with the ideal predictor always predicting 1_s for sequence s. On all benchmarks, partial update was consistently better than total update. The explanation lies in the way set S^* is built. Let us consider a sequence u in S^*. With a partial update, the 4-bit counter associated with u is updated only by the substream of u corresponding to occurrences of u has the longest matching sequence. On the other hand, with a total update, the 4-bit counter is updated even when there exist a matching sequence s longer than u. By construction of S^*, $1_s = 0_u$, which means that the counter associated with u is more likely to evolve toward the wrong sign when u is not the longest matching sequence.

Although we show results for 4-bit counters, we also experimented 2-bit counters and found that this generally increases the mispredict frequency, as predicted by the Markov chain model. As can be seen on Figure 3, 4-bit counters with a partial update permit approaching the ideal predictor. [2] It can be noticed on Figure 3 that, as $|T|$ increases, it

[2] On *crafty*, a 4-bit counter is slightly better than the ideal predictor. This probably comes from non-stationary branch behaviors, e.g., a branch changing direction in the midst of the PCF.

becomes difficult to come close to the ideal predictor. These extra mispredicts correspond mostly to low-frequency sequences, and may be called cold-start mispredicts. Cold-start mispredicts come from sequences not being used on their first occurrence and from 4-bit counters training time. In the remaining, when using 4-bit counters instead of the ideal prediction 1_s, partial update is implicit.

4.3 Conflicts between sequences

In real predictors, sequences are mapped onto table entries through hash functions. This generates conflicts between sequences. One may use a certain degree of associativity to remove conflicts. However, associativity is seldom used in conditional branch predictors, probably because it is not worth the extra complexity. We see two explanations. When the table is smaller than $|S^*|$, it is often possible to fill most table entries with useful sequences, even if these are not the most useful ones. Also, when two sequences from S^* are in conflict, it is sometimes possible to substitute a longer sequence for one of these.

The hash function we used in our experiments is an extension of *gshare* hash function [11]. It is based on $F(g, m)$ which folds a bit vector g onto m bits by a bitwise XOR (symbol $\oplus$) of groups of m consecutive bits. We define a hash function $H(s, m)$ onto m bits as follows. Sequence $s \in S(n)$ is represented by a branch word address a and a $n-1$ bits global history g. We append the sequence length to g, defining $g' = g \times 2^6 + n$. We define a m-bit value $h = F(g, m) \oplus 2F(g, m-1)$. If $n \leq m$, we define $h' = h \times 2^{m-n}$ (the $m-n$ most significant bits of h are null), otherwise $h' = h$. We take the m least significant bits of a and obtain a'. Finally, $H(s, m) = a' \oplus h'$.

To evaluate the impact of conflicts, we process the PCF twice. During the first pass, we build the tree and compute frequencies. The mispredict frequency is computed during the second pass. A difference with the ideal predictor is that we allow the content of T to change dynamically. The prediction is still given by the longest matching sequence s in T and its most frequent successor 1_s. However, at any time, we may replace certain sequences. We maintain a table with 2^m entries, where each entry can hold one sequence. At any time, T is the set of sequences that are in the table. After predicting the successor B_{j+1} of block B_j, we try to store sequences in the table according to the following allocation policy, for which we define two variants *alloc*+ and *alloc*− :

```
u = B_j
FOR n ∈ L, increasing
    s = B_{j-n+1} ··· B_j
    IF (s ∈ T)
        u = s
    ELSE IF (1_s ≠ 1_u)
        alloc+
            store s in entry H(s, m)
            u = s
        alloc−
            v = sequence stored in entry H(s, m)
            IF f(s.1_s) − f(s.0_s) ≥ f(v.1_v) − f(v.0_v)
                store s in entry H(s, m)
                u = s
```

Figure 4 shows the mispredict frequency in function of 2^m for the *alloc+* and *alloc−* allocation policies, compared with the ideal $m(T)$ in function of $|T|$. Results of our experiments indicate that using $H(s,m)$ with *alloc−* permits approaching the ideal predictor for moderate values of 2^m. On *gcc*, the ideal predictor was even outperformed, probably because of temporal locality. [3] For large values of 2^m, associativity may help a little. As can be seen on Figure 4, *alloc−* outperforms *alloc+*, because allocation is done more selectively. We observed a similar result on the other benchmarks. The *alloc+* policy, on the other hand, does not require information on sequence frequencies. It just requires the knowledge on the most frequent successor of a sequence.

4.4 Getting rid of oracle knowledge

Experiments in Section 4.2 have shown that a 4-bit counter is effective at guessing the most frequent successor of a sequence, provided we use a partial update. Experiments in Section 4.3 have shown that we can store sequences in a direct-mapped table without overly suffering from conflicts, but we need to know the most frequent successor of sequences. Naturally, the next step consists in combining 4-bit counters with the *alloc+* policy so that we no longer rely on oracle knowledge. However, there is a complication. The *alloc+* policy assumes, upon deciding whether or not to store a sequence s in the table, that 1_s is kown. If sequence s was stored somewhere, we would associate a 4-bit counter with it. Actually, this could be the case if the predictor under study was backed by a larger predictor, as proposed in [7]. For

[3] The ideal predictor cannot exploit temporal locality because the content of T is fixed for the whole PCF, hence a reason to work with PCFs that are not too long.

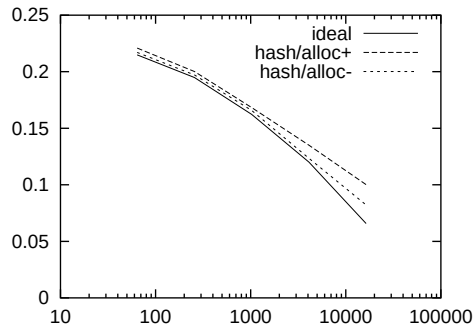


Figure 4: Example PCF : mispredict frequency in function of 2^m when using hash function $H(s,m)$ with the *alloc*+ and *alloc*− allocation policies, compared with the ideal $m(T)$ in function of $|T|$.

this study however, we assume that this is not the case. When a sequence is missing, we have no information on it and must rely on a heuristic. We introduce the *allocate-on-mispredict* allocation policy (AOM), which we define as follows :

$u \in S(n_{match})$ = longest matching sequence
IF mispredict
  FOR $n \in L, n > n_{match}$
    $s = B_{j-n+1} \cdots B_j$
    store s in entry $H(s,m)$

In the ideal case, *mispredict* means $1_u \neq B_{j+1}$. This allocation policy is an extension of that used in the YAGS predictor [4]. Figure 5 compares the mispredict frequency of AOM against that of *alloc*+. "AOM 4bc" denotes the same predictor as "AOM" but using a 4-bit counter instead of 1_u for the prediction. In this case, upon storing a sequence, its counter is initialized according to the branch outcome, i.e., state 0 if the branch is taken, state -1 otherwise. The degradation introduced by AOM is more pronounced on certain benchmarks. However, on all benchmarks, the biggest degradation comes from the use of 4-bit counters. As can be observed on Figure 5, the magnitude of the degradation is more important than one would expect from the analysis of Section 4.2.

In Section 4.2, the set T of sequences stored in the table was fixed. If sequences are encountered a sufficient number of times in the PCF, the counters have enough training and provide accurate predictions. Now, when a sequence is evicted from T, information on this sequence is lost. When the sequence reenters T at some time in the future, the 4-bit

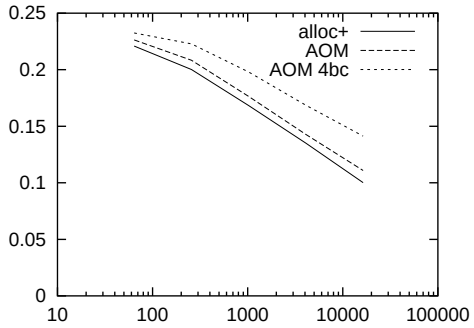


Figure 5: Example PCF : mispredict frequency in function of 2^m for the AOM allocation policy, against *alloc+*.

counter associated with the sequence is reinitialized. A 4-bit counter is asymptotically close to optimal, but the first few predictions are not very accurate (especially with the AOM policy, which often initializes counters with the "wrong" direction). Without knowledge of 1_s, we must rely on heuristics for trying to alleviate the problem. Through trial and error, we found a heuristic that seems to work well on average, although it does not solve the problem completely. This heuristic relies on extra information maintained dynamically. We associate a bit $useful(s)$ to each sequence $s \in T$, and we associate 2-bit up-down saturating counters $meta(B, n)$ to each $B \in \mathcal{B}$ and for each $n \in L$. We denote $p(u)$ the prediction given by the 4-bit counter associated with a sequence u. The heuristic is described on Figure 4.4. We denote this predictor "AOM 4bc+". It should be noted that $meta(B_j, n)$ is updated in a way similar to the meta-predictor introduced in [11]. The $meta(B_j, n)$ counter tries to identify cases where it is beneficial to use longer sequences. A small $m(n)$ often indicates that there is a lot of correlation between branches. In that case, $meta(B_j, n)$ will often be positive, which leads to an "aggressive" initialization of 4-bit counters according to B_{j+1} (state 0 if the branch ending B_j is taken, state -1 otherwise). On the other hand, if there is little correlation, it is safer to initialize 4-bit counters with $p(B_j)$. Figure 7 compares the mispredict frequency of AOM 4bc+ against that of AOM and AOM 4bc. On a majority of benchmarks, "AOM 4bc+" is very close to "AOM". However, there is room for improvement on certain benchmarks (*gzip*,*vpr*, *twolf*).

4.5 A predictor that could be implemented

Predictor AOM 4bc+ introduced in the previous section no longer relies on oracle information. However, further degradations are necessary for an implementation in hardware.

```
u ∈ S(n_match) = longest matching sequence
IF p(u) ≠ B_{j+1}
    alloc_failed = TRUE
    FOR n ∈ L, n > n_match
        s = B_{j-n+1} ··· B_j
        v = sequence stored in entry H(s,m)
        IF useful(v) = 0
            store(s,n)
            alloc_failed = FALSE
    IF alloc_failed
        n = choosen randomly in L, n > n_match
        s = B_{j-n+1} ··· B_j
        store(s,n)
IF p(u) ≠ p(B_j)
    IF p(u) = B_{j+1}
        increment meta(B_j, n_match)
        useful(u) = 1
    ELSE
        decrement meta(B_j, n_match)
        useful(u) = 0
update 4-bit counter associated with u
```

```
DEF store(s,n)

store s in entry H(s,m)
useful(s) = 0
IF meta(B_j, n) ≥ 0
    initialize 4-bit counter
    according to B_{j+1}
ELSE
    initialize 4-bit counter
    according to p(B_j)
```

Figure 6: Algorithm for "AOM 4bc+". Notation $p(u)$ represents the prediction given by the 4-bit counter associated with a sequence u.

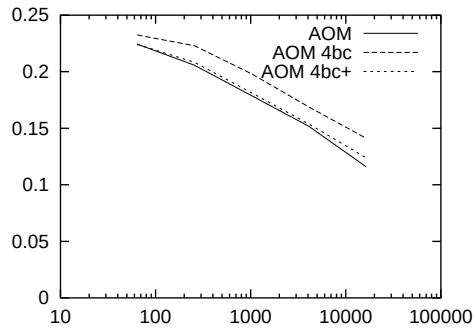


Figure 7: Example PCF : mispredict frequency in function of 2^m for AOM 4bc+ vs. AOM and AOM 4bc

Banking. Table T should be banked so that finding the longest matching sequence be done in parallel instead of sequentially. Given $L = \{6, 11, 21, 41\}$, we assume four equally-sized banks for T. Each bank has 2^{m-2} entries and is indexed with $H(s, m-2)$. For simplifying the logic, we assume that each sequence length is mapped to a dedicated bank, e.g., banks $0, 1, 2, 3$ correspond to $n = 6, 11, 21, 41$ respectively. Figure 8 shows the impact of banking on the example PCF. As can be observed, banking has little impact. We did not expect this result, because we thought that allocating a fixed fraction of the overall table space to each sequence length would degrade the predictor. Actually, this is not the case. In order to understand this phenomenon, we returned to the *alloc+* and *alloc−* predictors studied in Section 4.3 and measured, at the end of simulation, the proportion of each sequence length stored in T. On the example PCF, and with *alloc−*, length $n = 41$ contributes very little, which is what we expected from experiments in Section 4.1. However, with *alloc+*, sequence lengths are roughly equally distributed. In other words, the degradation we expected by banking had already happened before. The equipartition of the table space among the different sequence lengths is inherent to the allocation policy. It comes from the lack of knowledge on sequence frequencies.

Tag widths. Instead of storing the whole identifier of a sequence in table T, we store a tag obtained by applying a hash function H' on the sequence. Considering a fixed number of entries, this is obviously a degradation. With t tag bits, if the hash function is well chosen, the probability that two tags are aliased is 2^{-t}. So, as noted in [4], the tag does not need to be long. One should should H' so that $H(s, m-2)$ and $H'(s, t)$ are not correlated. For our experiments, we took $H'(s, t)$ as the t least significant bits of $a \oplus (a/8) \oplus F(g, t)$ (cf. Section 4.3). Figure 9 shows the impact of tag width on the example PCF. We found that with $t = 8$ tag bits, the degradation is negligible.

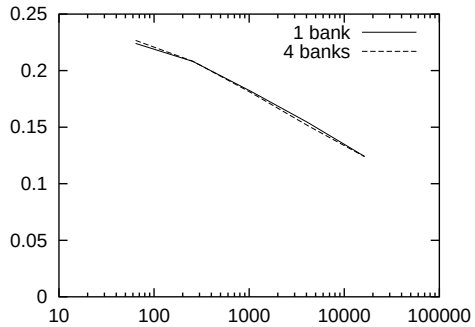


Figure 8: Example PCF : mispredict frequency in function of 2^m for 1 bank against 4 banks.

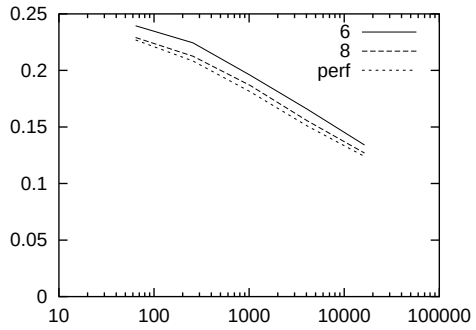


Figure 9: Example PCF : mispredict frequency in function of 2^m for $t = 6$ and $t = 8$ tags bits, against the perfect case (full tag).

benchmark	64 Kbits		256 Kbits		1 Mbits	
	2bcgskew	AOM 4bc+	2bcgskew	AOM 4bc+	2bcgskew	AOM 4bc+
164.gzip	0.092	0.082	0.092	0.082	0.092	0.082
175.vpr	0.079	0.073	0.078	0.069	0.076	0.065
176.gcc	0.042	0.039	0.035	0.028	0.030	0.023
186.crafty	0.041	0.036	0.034	0.028	0.030	0.026
197.parser	0.050	0.045	0.044	0.042	0.040	0.041
300.twolf	0.138	0.117	0.128	0.109	0.125	0.101
go	0.182	0.164	0.134	0.120	0.106	0.082

Table 1: Mispredict rate of AOM 4bc+ vs. *2bcgskew*

Evaluation. Given a limited storage capacity (measured in bits), the AOM 4bc+ predictor achieves a good utilization of this capacity. We compared the predictor with *2bcgskew*, which is known to achieve a low mispredict rate for a fixed capacity. In AOM 4bc+, a part of the overall capacity is used to record information associated with $B \in \mathcal{B}$. We assume this information is stored in a "bimodal" table indexed as usual, i.e., with the least significant bits of the address of the branch ending block B. Overall, with table T, we have 5 banks. We assume that all 5 banks have the same number of entries, i.e., 2^{m-2} entries. Each entry of the "bimodal" table holds one 4-bit counter and four $meta(B,n)$ 2-bit counters, for a total of 12 bits per entry. Each entry of table T holds one 8-bit tag, one 4-bit counter, and one bit $useful(s)$, for a total of 13 bits. The total number of bits is $2^{m-2} \times 12 + 2^m \times 13 = 2^{m+4}$ bits, of which half are used in tags.

For *2bcgskew*, we used the publicly available simulator [1]. It implements an optimized *2bcgskew* that is evaluated in [13]. We did not modify it. N_{bits} being the total storage capacity in bits, two tables use a global history length of $\log_2 N_{bits} - 11$, the third table uses length $4 \times (\log_2 N_{bits} - 11)$, and the fourth one uses length $8 \times (\log_2 N_{bits} - 11)$. We simulated predictor capacities of 64 Kbits, 256 Kbits and 1 Mbits. For AOM 4bc+, this corresponds to 1024, 4086, and 16384 entries per bank respectively (i.e., $m = 12, 14, 16$). We ran each benchmark for 100 millions instructions, after skipping the first 100 millions ones. Mispredict rates (i.e., number of mispredicts divided by number of conditional branches) are reported on Table 1. The mispredict rate of AOM 4bc+ is lower than that of *2bcgskew*, except for *197.parser* with 1 Mbits of storage.

4.6 Further degradations ?

The AOM 4bc+ predictor is more complex to implement than *2bcgskew*. Complexity on the update path ($meta(B,n)$, $useful(s)$, ...) can be tolerated to some extent, but complexity on the prediction path increases the mispredict penalty. We do not think that hash functions are really a problem. The hash functions we used can be simplified when designing a predictor with definite size and sequence lengths. Yet, for obtaining a prediction, we read five prediction bits and four 8-bit tags, i.e., a total of 37 bits. These 37 bits are then reduced to one final prediction. The logic to implement this reduction can incur a delay of several cycles. One could decrease complexity by shortening the tags. But tags must have a minimum width to be effective (Figure 9), so this is not really a solution.

4.7 Hints for future improvements

An straightforward way to improve the predictor on certain benchmarks would be to consider a maximum sequence length greater than $n_{max} = 41$. For example, we could add a fifth bank on table T, associated with a sequence length longer than 41, or keep four banks but distribute sequence lengths more sparsely. Yet, for a fixed n_{max}, there is a gap between the mispredict rate of AOM 4bc+ and that of the ideal predictor we started from. In particular, we have noted in Section 4.3 that the lack of knowledge on sequence frequencies incurs a significant degradation (*alloc*− vs. *alloc*+). A possible direction to explore would be the

use of quotas that would give less table space to longer sequences, as long sequences are less useful than short ones for a majority of benchmarks. This may require to bank table T differently (Section 4.5). Another important degradation was observed between AOM and AOM 4bc. It comes from the lack of knowledge of 1_s when bringing sequence s in the table. The introduction of $meta(B, n)$ and $useful(s)$, that led to AOM 4bc+, attenuated the degradation, but it is still significant on certain benchmarks. More efficient ways to palliate the problem may exist.

5 Conclusion

Our first motivation for using frequency was its conceptual simplicity. Then we found during our study that, had we not used frequency, we may have missed important phenomena associated with 4-bit counters, in particular partial vs. total update and AOM vs. AOM 4bc. These phenomena were not obvious to us beforehand, and the method was instructive in that respect.

The method proposed is not intended to replace the usual one, but to complement it. We did not choose the ideal predictor and the degradation path arbitrarily, but with a direction in mind, based on our familiarity with tag-based predictors.

It is not clear whether our ideal predictor can be used as a model for tag-less predictors. The advantage of tags is that, if they are wide enough, aliasing is negligible. Tag-less predictors, on the other hand, try to tolerate aliasing. For example, *2bcgskew* combines a meta-predictor and a majority vote. Another example is COLT fusion [10], which provides a general way to build tag-less predictors. Using selection information more compact than tags should decrease the predictor delay, and also decrease the number of bits per table entry, which allows to have more entries for the same total storage capacity (though it has not yet been demonstrated that tags are not space efficient, see Table 1).

References

[1] The 2bcgskew simulator. http://www.irisa.fr/caps/ people/seznec/2bcgskew.html.

[2] I.-C.K. Chen, J.T. Coffey, and T.N. Mudge. Analysis of branch prediction via data compression. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[3] J.G. Cleary and I.H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, April 1984.

[4] A.N. Eden and T. Mudge. The YAGS branch prediction scheme. In *Proceedings of the 31st Annual International Symposium on Microarchitecture*, 1998.

[5] E. Federovsky, M. Feder, and S. Weiss. Branch prediction based on universal data compression algorithms. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, 1998.

[6] W. Feller. *An introduction to probability theory and its applications*, volume 1. Wiley, 1950.

[7] D.A. Jiménez, S.W. Keckler, and C. Lin. The impact of delay on the design of branch predictors. In *Proceedings of the 33rd Annual International Symposium on Microarchitecture*, 2000.

[8] D.A. Jimenéz and C. Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems*, 20(4), November 2002.

[9] J.F.K. Lee and A.J. Smith. Branch prediction strategies and branch target buffer design. *IEEE Computer*, January 1984.

[10] G.H. Loh and D.S. Henry. Predicting conditional branches with fusion-based hybrid predictors. In *Proceedings of the 11th Conference on Parallel Architectures and Compilation Techniques*, 2002.

[11] S. McFarling. Combining branch predictors. Technical note TN-36, DEC WRL, June 1993.

[12] R. Nair. Dynamic path-based branch correlation. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, 1995.

[13] A. Seznec. Redundant history skewed perceptron predictors. PI-1554, IRISA, September 2003.

[14] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides. Design tradeoffs for the Alpha EV8 conditional branch predictor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, 2002.

[15] J.E. Smith. A study of branch prediction strategies. In *Proceedings of the 8th Annual International Symposium on Computer Architecture*, 1981.

[16] J. Stark, M. Evers, and Y.N. Patt. Variable length path branch prediction. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1998.

[17] R. Thomas, M. Franklin, C. Wilkerson, and J. Stark. Improving branch prediction by dynamic dataflow-based identification of correlated branches from a large global history. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2003.

[18] T.-Y. Yeh and Y.N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, May 1992.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399