An Alternative TAGE-like Conditional Branch Predictor

Pierre Michaud

Inria, Univ Rennes, CNRS, IRISA
pierre.michaud@inria.fr

May 24, 2018

Abstract

TAGE is one of the most accurate conditional branch predictors known today. However, TAGE does not exploit its input information perfectly, as it is possible to obtain significant prediction accuracy improvements by complementing TAGE with a statistical corrector using the same input information. This paper proposes an alternative TAGE-like predictor making statistical correction practically superfluous.

1 Introduction

The branch predictor is the keystone of modern superscalar microarchitectures. Reducing the number of branch mispredictions is a relatively simple way to increase performance and simultaneously decrease energy consumption.

Research in branch prediction has been a decades-long effort, largely focused on predicting the direction of conditional branches, as this is generally the greatest source of mispredictions. Since 2006, research in branch prediction has progressed only slowly. A plausible reason is that modern branch predictors might be close to a prediction accuracy limit. However, this is not certain either, as the only mathematically proven limit is zero misprediction, which is still far from being attained.

Today, a highly accurate conditional branch predictor based on the published state of the art is either TAGE-like [69], perceptron-based [35, 29], or a combination of those [66, 30]. Both types of predictors have distinct advantages: TAGE exploits the limited predictor storage very efficiently, whereas perceptron-based predictors can easily combine different sorts of input information.

This paper focuses on TAGE-like predictors. Except for the first branch prediction championship, which was held in 2004 before TAGE was introduced [2], all four subsequent championships were won by predictors based on TAGE [3, 4, 5, 6]. It is not known with certainty which commercial processors implement a TAGE-like predictor, as this information is rarely disclosed. Admittedly, a PPM-like predictor is implemented in the IBM zEC12 [1] and a TAGE predictor is implemented in the Phytium Mars [82] and in the IBM POWER9 [26].

Although TAGE is a very accurate predictor, it does not exploit its input information perfectly, as significant prediction accuracy improvements are obtained by complementing TAGE with a *statistical corrector* using the same input information [63]. The statistical corrector, even small, makes the whole predictor more complex. More importantly perhaps, this imperfection of TAGE revealed by the statistical corrector is a hint that our understanding of TAGE-like predictors is incomplete.

This paper proposes an alternative TAGE-like predictor, called BATAGE, making statistical correction practically superfluous. BATAGE has the same global structure as TAGE but uses a different tagged-entry format and different prediction and update algorithms.

Section 2 provides the historical and technical context for situating the contributions of BATAGE. Section 3 analyzes the *cold-counter* problem, which is the main cause for TAGE needing statistical correction. Section 4 proposes to replace the conventional up/down counter in TAGE with two counters counting separately the taken and not-taken occurrences. A new method is proposed for estimating prediction confidence using Bayesian probabilities. Section 5 describes the BATAGE predictor and its features, including

- a new prediction automaton called *dual-counter*,
- prediction and update algorithms exploiting the dual-counter and Bayesian confidence estimation,
- a new method called Controlled Allocation Throttling (CAT) for reducing the allocation frequency.

Finally, Section 6 provides an experimental evaluation of BATAGE.

2 A short history of conditional branch prediction

BATAGE is derived from TAGE, which is the result of decades of research in branch prediction. This section provides a short history of conditional branch prediction. Its purpose is to furnish the historical and technical context for situating the contributions of BATAGE.

Figure 1 illustrates the improvements in prediction accuracy over the years. It shows the average MPKI (mispredictions per 1000 instructions) over the CBP 2016 traces for various branch predictors since 1993. Appendix A provides a detailed legend of Figure 1.

2.1 The eighties

The *bimodal* predictor, that is, a table of up/down saturating counters accessed with a hash of the branch address, was introduced by James E. Smith in the early eighties [70].[1]

In 1984, Johnny Lee and Alan Jay Smith proposed a branch prediction method consisting of branch history bit vectors stored in the Branch Target Buffer (BTB) entries and used for accessing a prediction table trained offline with representative workloads [40]. They were focusing on understanding the prediction accuracy limits of this method. They did not describe a hardware implementation. Although Lee and Smith mentioned in passing the possibility of having a global branch history (using today's terminology), their study focused on per-branch history bit vectors.

2.2 Two-level prediction

In a seminal paper published in 1991, Yeh and Patt proposed a branch predictor derived from Lee and Smith's method but using online instead of offline training [78]. They introduced the *Pattern History Table* (PHT), a prediction table holding 2-bit up/down counters and accessed with the history bit vector of the branch being predicted. They called this predictor *Two-Level Adaptive*. In the 1991 paper, Yeh and Patt considered only per-branch (aka local) history bit vectors. The first papers describing a global-history predictor were published in 1992 by Yeh and Patt [79] and by Pan, So and Rameh [50]. In a global-history predictor, there is a

[1] The 2-bit automaton used in the branch predictor of the S-1 computer built at the Lawrence Livermore Laboratory during the seventies was not an up/down counter [40]. The S-1 automaton is generally less accurate than the 2-bit up/down counter [51, 78, 49].

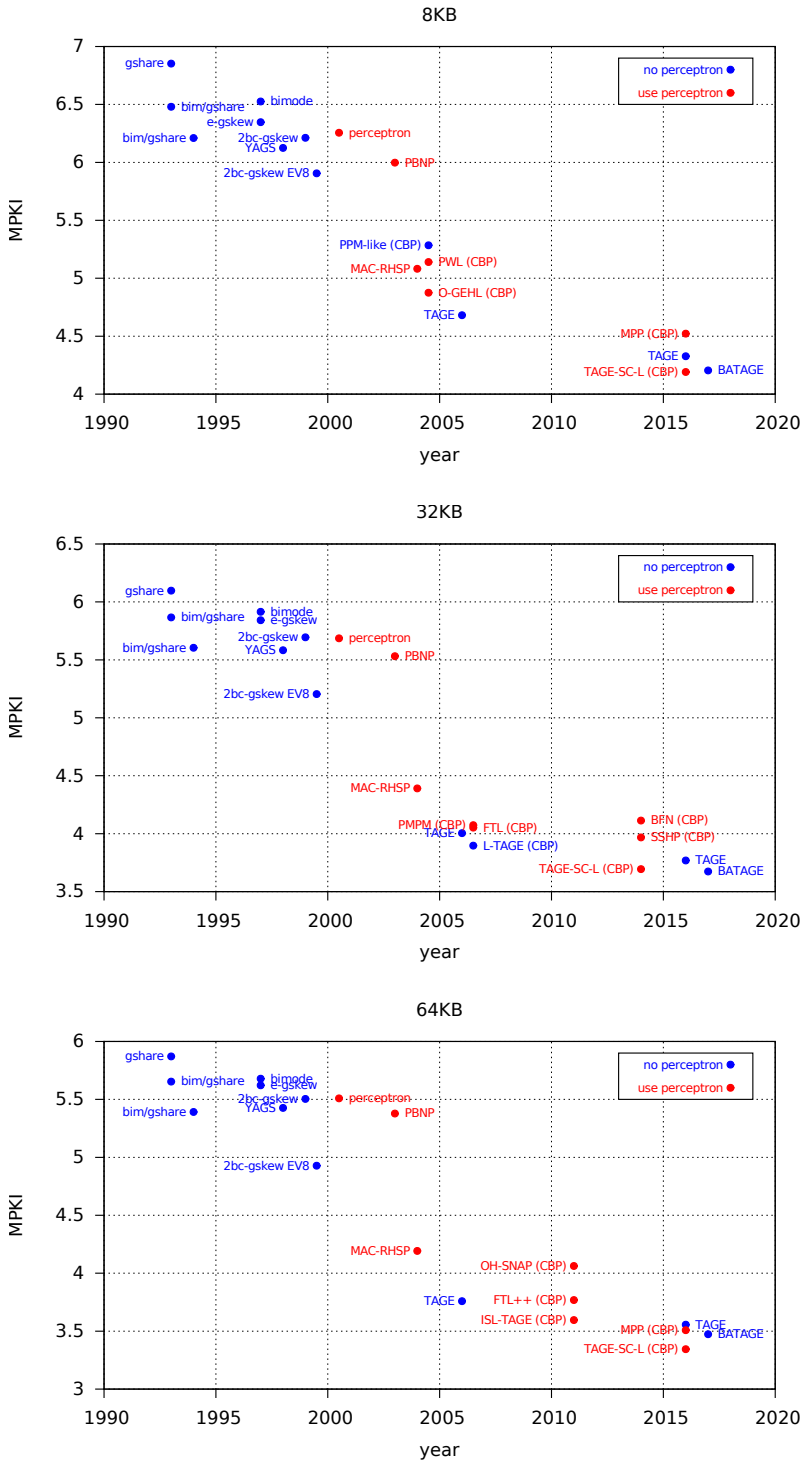

Figure 1: Average number of mispredictions per 1000 instructions (MPKI) for various conditional branch predictors on the CBP 2016 traces for 8KB, 32KB and 64KB storage budgets (see Appendix A).

global history register shared by all the branches. Yeh and Patt generalized the label "Two Level", including global-history predictors as a special case [79, 80]. This first wave of two-level schemes culminated with the *gshare* global-history predictor introduced by McFarling [42]. For several years, *gshare* was the reference conditional branch predictor for its simplicity and high prediction accuracy.[2]

2.3 Meta-prediction and de-aliasing techniques

In 1993, McFarling introduced *meta-prediction*, a method for combining several different prediction methods, leading to *hybrid* predictors more accurate than gshare [42]. As Figure 1 shows, the *bimodal/gshare* hybrid predictor of 1993, and the improved version of 1994 [44], have an MPKI significantly lower than gshare. More research on hybrid predictors was published in the following years [9, 10, 19, 18, 41].

During the same period, some papers studied the *aliasing* problem stemming from the limited branch predictor size [74, 81, 53]. There was a wave of studies trying to overcome the aliasing problem [7, 72, 47, 39, 17]. This line of research culminated with the branch predictor of the Alpha EV8 processor [67], derived from Seznec's 2bc-gskew predictor [68].

The MPKIs of some emblematic "de-aliased" predictors are shown in Figure 1 (1997-1999). These predictors are indeed more accurate than gshare. But compared against a well-tuned bimodal/gshare hybrid, the accuracy of de-aliased predictors is not much better, if better at all. The discrepancy between Figure 1 and the progresses claimed in the original papers is perhaps due, in part, to the CBP 2016 traces having different characteristics from the benchmarks used at the time. Still, the use of McFarling's meta-predictor as a de-aliasing technique was certainly overlooked back then. Nevertheless, research on de-aliased predictors improved our understanding of branch prediction. Eventually, the lessons from de-aliased predictors led to significant accuracy gains with the branch predictor of the Alpha EV8 processor (2bc-gskew-EV8 in Figure 1).

2.4 The Perceptron

In 1999-2000, two research teams, independently, started exploring the use of artificial neural networks for branch prediction [77, 35]. In particular, Jiménez and Lin's perceptron predictor had an important impact, not so much for the demonstrated prediction accuracy as for the completely different, thought-provoking branch prediction algorithm [35, 37]. Remarkably, the perceptron was the first branch predictor that could take advantage of a global history longer than 60 branches. It started the trend toward very long global histories.[3]

In their 2001 paper, Jiménez and Lin emphasized a limitation of the original perceptron: *linear separability*. In the following years, several studies tackled linear separability [31, 56, 58, 32]. In particular, Seznec's RHSP was the first perceptron-based predictor to demonstrate a prediction accuracy substantially better than that of de-aliased predictors [56, 58]. The hardware complexity of perceptron-based predictors was greatly reduced in 2004 when, inspired from Seznec's MAC-RHSP [58], Tarjan and Skadron with the Hashed Perceptron [75, 76] and Seznec with the GEHL predictor [57, 59], decoupled the perceptron width from the global history length.

[2] On the CPB 2016 traces, the *average* MPKI of an 8KB gshare is about 35% lower than that of a large bimodal predictor.

[3] A recent study explains in detail how to implement a long global history [52].

2.5 TAGE

In 1996, Chen et al. considered adapting Prediction by Partial Matching (PPM), a data compression technique [13], to branch prediction [12]. More specifically, they applied PPM to local-history two-level prediction, assuming a PHT with as many entries as the number of distinct local-history values of all possible lengths up to a maximum length. Extra bits were stored in the PHT to detect cold PHT entries [11]. Another research team confirmed the potential of PPM for branch prediction, but without considering hardware implementation constraints [20]. PPM was also proposed for indirect jump prediction [38].

Conventional BTBs feature tags [71, 25, 40], and in 1997-1998, when researchers started exploring more sophisticated indirect jump predictors, the use of tags was an obvious option [8, 15, 14]. However, the use of tags in conditional branch predictors was counterintuitive, as this meant much fewer 2-bit up/down counters at equal storage. The fact that tags are a cost-effective option for conditional branch prediction was discovered by McFarling with the serial-BLG predictor [43] and by Eden and Mudge with the YAGS predictor [17].

In 2001, I introduced *gtags*, an approximation of PPM consisting of several cascaded YAGS using increasing global history lengths[4] [46], somewhat similar to Driesen and Hölzle's multi-cascaded indirect jump predictor [16], but for conditional branches. I observed that prediction accuracy increases with the number of distinct global history lengths. However, at the time, it seemed difficult to implement a cost-effective *gtags* with more than three tables. In particular, I noticed that freshly allocated entries sometimes generated many mispredictions, a problem that I call today the *cold-counter* problem (see Section 3). In 2004, I submitted to the first branch prediction championship a predictor called *PPM-like* [45], which was a 5-table *gtags* but with two key improvements (Figure 2). The first improvement was the introduction of a u bit in each tagged entry for estimating the usefulness of that entry, in order to better manage the limited predictor storage. The second improvement was the introduction of an m bit in tagless entries for controlling the initialization of the up/down prediction counter upon allocation of a tagged entry, in order to mitigate the cold counter problem. The PPM-like predictor was the first predictor not derived from the perceptron with a prediction accuracy substantially better than that of de-aliased predictors (Figure 1, 8KB budget).

In 2005, Seznec improved the PPM-like predictor quite substantially, which led to TAGE [69]. Unlike the PPM-like predictor, which uses a global history of branch directions, TAGE uses a path history [81, 48]. The key features distinguishing TAGE from PPM-like are:

- TAGE considers not only the longest matching path (*pred*), but also the second-longest matching one (*altpred*).
- A tiny meta-predictor selects the final prediction between *pred* and *altpred* depending on the state of the *pred* up/down counter. This is for tackling the cold counter problem. Upon tagged entry allocation, the up/down counter is always initialized according to the branch outcome (TAGE has no m bit).
- Each tagged entry contains a u counter (generalization of the u bit).
- The *pred* entry is considered useful when it prevents the occurrence of a misprediction.
- The u counters are periodically decremented *en masse* so that dead entries are not locked forever.
- TAGE allocates a single entry per misprediction.

[4] In 2001, I quickly discarded the use of arithmetic progressions for the global history lengths and used a geometric progression instead (8,16,32,64) [46]. The fact that a geometric progression is often close to optimal was discovered and emphasized by Seznec a few years later [59, 69].

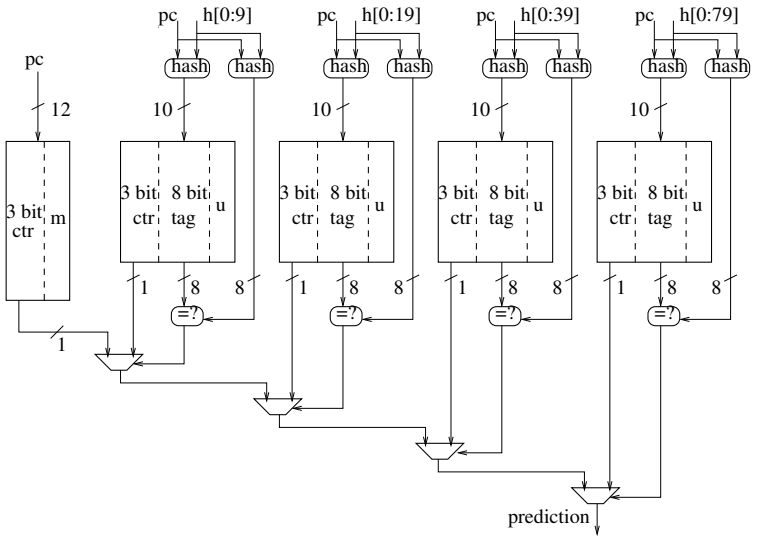

Figure 2: The PPM-like predictor of CBP 2004 [2, 45].

All four branch prediction championships between 2006 and 2016 were won by André Seznec with a TAGE based predictor. Figure 1 shows two TAGE versions, the original 2006 version, and the latest 2016 version (see Appendix A). Seznec did a few changes over the years (in particular, he improved the management of u counters in 2011 [63]), sometimes depending on the storage budget considered. Nevertheless, the fundamentals of TAGE have not changed. The better accuracy of TAGE 2016 over TAGE 2006 is largely due to TAGE 2016 having more banks (and path lengths) and using bank interleaving, where a tagged bank is shared by several path lengths [62, 66].

2.6 Statistical correction

Gao and Zhou noticed that, in a PPM-like predictor, the longest match is not always the most accurate [22], corroborating and generalizing what Seznec observed for TAGE [69]. The PMPM predictor they submitted to CBP 2006 combined some GEHL features (geometric path lengths, perceptron algorithm, adaptive threshold) with some TAGE features (tags, u counters) [22]. However, they did not succeed in outperforming a well-tuned TAGE [3, 61].

In 2006, Seznec noticed that, for huge storage budgets, and using the same input information, GEHL outperforms TAGE [60]. Later, he found that this was due to TAGE behaving suboptimally on hard-to-predict branches, that is, weakly-biased branches with little or no global correlation. He proposed to complement TAGE with a statistical corrector, and found that a perceptron-inspired predictor such as GEHL is a cost-effective statistical corrector for TAGE [63]. Seznec won the last three championships with this kind of predictor, called TAGE-SC [62, 65, 66].

Prediction accuracy has improved a lot since gshare: as shown in Figure 1, the 64KB TAGE-SC-L of CBP 2016 has an *average* MPKI 43% lower than a 64KB gshare.[5]

[5]It should be recalled that the CBP rules do not limit predictor complexity besides a fixed storage budget. The 64KB TAGE-SC-L of CBP 2016 features a TAGE with 30 interleaved tagged banks and a statistical corrector with 20 weight tables and 3 local-history tables [66].

3 The cold counter problem

The cold counter problem is specific to PPM/TAGE-like predictors. Yet, although some features were introduced in the PPM-like predictor and in TAGE to reduce it, the cold-counter problem was neither named nor emphasized. I briefly mentioned *some* aspects of the problem in 2001 [46], and it was only implicit in subsequent papers [45, 69]. The paper that introduced statistical correction seems to downplay the importance of the meta-predictor in TAGE (called USE_ALT_ON_NA) but does not explain that it is the statistical corrector that renders the meta-predictor practically superfluous [63]. The importance of the meta-predictor, which is real without the statistical corrector, is stated explicitly only in the source code of TAGE-SC-L in 2016 [6]. It is safe to say that the cold-counter problem has long been underestimated and did not receive the attention it deserves. This section provides a mathematical analysis of the cold-counter problem.

Initially, an up/down counter has no information about the branch (or path) it is trying to predict. After one occurrence of the branch, the up/down counter provides a prediction which is generally better than a random guess. In fact, if the branch is biased very strongly toward one direction, an up/down counter trained with a single occurrence of the branch is close to an ideal predictor for that branch. However, if the branch is weakly biased, it may take many occurrences of the branch for the counter to guess the branch bias with high confidence (assuming the counter is wide enough to record that amount of branch history). The number of occurrences necessary to train a wide-enough counter can be surprisingly large, as can be seen by modeling a single branch as a Bernoulli process, with probability q for the branch to be taken: the probability $M(n)$ to mispredict the branch with a (wide-enough) up/down counter previously trained with n occurrences of the branch is

$$\text{for } n \text{ odd}, \quad M(n) \;=\; M(n+1) \;=\; \sum_{k=0}^{\frac{n-1}{2}} \binom{n}{k} q^{k+1}(1-q)^{n-k} \;+\; \sum_{k=\frac{n+1}{2}}^{n} \binom{n}{k} q^{k}(1-q)^{n-k+1}$$

where it is assumed that the counter is initialized in state 0 when the first occurrence is taken, in state -1 when it is not taken.[6]

Figure 3 shows $M(n)$ as a function of n, for $q = 0.1$ and for $q = 0.3$. The misprediction probability decreases as we accumulate knowledge about the branch and, in the limit, approaches the optimal misprediction probability $\min(q, 1-q)$. However, convergence is slow. For prediction accuracy to be within 5% of the optimal accuracy, it takes 8 occurrences of the branch with $q = 0.1$, and 20 occurrences with $q = 0.3$.

PPM/TAGE-like predictors suffer from the cold counter problem because they allocate tagged entries on mispredictions, resetting the up/down counter of newly allocated entries. Worse, if there is no global correlation, the up/down counter tends to be initialized in the wrong direction. In the PPM-like predictor, the cold counter problem was somewhat mitigated by trying to initialize the up/down counter intelligently. TAGE provided a more effective solution to the cold counter problem with the meta-predictor selecting between the longest and second-longest matching paths. Nevertheless, TAGE does not solve the cold counter problem completely. This is why Seznec obtained significant accuracy gains by combining TAGE with a GEHL statistical corrector, as perceptron-based predictors such as GEHL barely suffer from the cold-counter problem.

[6] The fact that $M(n)$ decreases only on odd values of n can be understood intuitively as follows. The prediction from the counter is a majority vote over the past n occurrences. When n is odd, the counter cannot change its prediction upon the n+1-th occurrence unless the number of taken and not-take occurrences becomes equal, in which case the prediction is no better than a random guess.

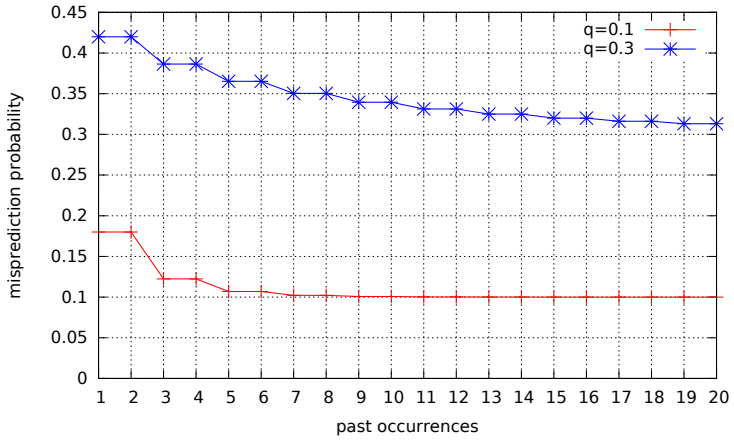

Figure 3: Illustration of the cold counter problem: misprediction probability for a wide-enough up/down counter on a branch having probability q to be taken, as a function of the number of past occurrences.

4 Bayesian confidence estimation

While complementing TAGE with a statistical corrector is a possible solution to the cold-counter problem, it increases the branch predictor complexity and the branch prediction delay. In this paper, I propose to solve the cold-counter problem without a statistical corrector.

There are two fundamental reasons why the cold-counter problem exists in TAGE:

1. TAGE allocates tagged entries on nearly every branch misprediction.

2. TAGE cannot estimate precisely the "temperature" of an up/down counter.

This section addresses the second point (the first point is addressed in Section 5.6).

Upon a tagged entry allocation, the counter is initialized in state 0 if the branch is taken, in state -1 if the branch is not taken. Subsequently, every time this entry is used to make a prediction, the counter is incremented or decremented depending on the branch direction. If we find the counter in state -4, we are sure that the counter has been updated at least 3 times after the initialization, so we know that the counter is relatively warm. However, if we find the counter is state -1, we do not know if the counter has just been initialized or has been updated multiple times. That is, we cannot distinguish between a cold counter and a weakly biased branch.

Therefore, I propose to replace the up/down counter with two counters n_1 and n_0 counting respectively the number of taken and not-taken occurrences. This way, the "temperature" of a tagged entry can be estimated precisely. Incidentally, the introduction of n_1 and n_0 also permits removing the u counter (explanation in Section 5.3). This change of tagged entry format, shown in Figure 4, is the foundation of BATAGE. Upon a tagged entry allocation, n_1 and n_0 are both reset, and either n_1 or n_0 is incremented according to the branch direction. On subsequent updates, n_1 or n_0 increase depending on the branch direction (ignoring for the moment the limited counter width). The prediction is given by the sign of $n_0 - n_1$.

I propose to tackle the cold counter problem by associating prediction confidence levels with values of n_1 and n_0. This way, when a tagged entry gives a low-confidence prediction, we can choose to ignore it and go to the next hitting tagged entry.

Figure 4: Tagged entry in TAGE (left) vs. BATAGE (right). The payload of a TAGE entry consists of an up/down counter (typically, 3 bits) and a u counter (1 or 2 bits). The payload of a BATAGE entry consists of two counters n_1 and n_0 (typically, 3 bits each) counting respectively the number of taken and not-taken occurrences.

4.1 Estimating branch predictability

In 1981, Smith observed that, when an up/down counter is saturated, this is an indication that the branch prediction provided by that counter is likely to be correct [70]. Smith's method can be generalized as follows. Let $x \in [-2^c, 2^c - 1]$ be the value of the up/down counter. A fine-grained confidence level can be obtained as $|2x + 1|$: the higher $|2x + 1|$, the more confident the prediction [64]. A coarse-grained confidence level can obtained by comparing $|2x + 1|$ with a threshold [36].

Defining confidence levels from two counters n_1 and n_0 is not as straightforward. Obviously, there exists many different possibilities. Defining the confidence level as $|n_1 - n_0|$ is definitely *not* what we want, as this would be practically equivalent to having an up/down counter. Instead, we would like to exploit the information that n_1 and n_0 provide and that is not available from an up/down counter.

For example, let us consider two predictors giving different predictions:

- **predictor A**: $n_1 = 2$, $n_0 = 0$ (predict *taken*)
- **predictor B**: $n_1 = 3$, $n_0 = 6$ (predict *not-taken*)

Given that sole information, which predictor is the most reliable? If we were using $|n_1 - n_0|$ as confidence level, we would select prediction B. However, there is nothing obvious here. Maybe we should trust predictor A instead, even though it is colder. In fact, the question is not well defined. To answer it, we must make some assumptions.

I propose to assume that a branch has a constant but unknown probability q to be taken for a given path, with q uniformly distributed in $[0, 1]$. This assumption is somewhat arbitrary. Nevertheless it does not introduce any bias toward *taken* or *not-taken* and it is consistent with the fact that we know nothing about branches but what we learn through n_1 and n_0. With this assumption, and using a Bayesian reasoning, it is possible to define confidence levels. Let us start from a uniform prior distribution $P_0(q) = 1$, and let us apply Bayes' theorem to obtain a posterior distribution $P(q|n_1, n_0)$:

$$P(q|n_1, n_0) = \frac{P(n_1, n_0|q) \times P_0(q)}{\int_0^1 P(n_1, n_0|t)P_0(t)dt} = \frac{P(n_1, n_0|q)}{\int_0^1 P(n_1, n_0|t)dt}$$

As $P(n_1, n_0|q) = \binom{n_1+n_0}{n_1} q^{n_1}(1-q)^{n_0}$, we obtain

$$P(q|n_1, n_0) = \frac{q^{n_1}(1-q)^{n_0}}{B(n_1 + 1, n_0 + 1)}$$

where $B(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt$ is the beta function. With the posterior distribution, we can obtain an estimate of the misprediction probability, which we denote $\widehat{m}$. If $n_1 < n_0$, the prediction is *not-taken*, and the misprediction probability is q, hence

$$\widehat{m} \;=\; \int_0^1 qP(q|n_1, n_0)dq = \frac{B(n_1 + 2, n_0 + 1)}{B(n_1 + 1, n_0 + 1)}$$

		n_1							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
n_0	**0**	0.50	0.33	0.25	0.20	0.17	0.14	0.12	0.11
	1	0.33	0.50	0.40	0.33	0.29	0.25	0.22	0.20
	2	0.25	0.40	0.50	0.43	0.38	0.33	0.30	0.27
	3	0.20	0.33	0.43	0.50	0.44	0.40	0.36	0.33
	4	0.17	0.29	0.38	0.44	0.50	0.45	0.42	0.38
	5	0.14	0.25	0.33	0.40	0.45	0.50	0.46	0.43
	6	0.12	0.22	0.30	0.36	0.42	0.46	0.50	0.47
	7	0.11	0.20	0.27	0.33	0.38	0.43	0.47	0.50

Table 1: Estimator $\widehat{m}$ as a function of n_1 and n_0 (formula (1)).

If $n_1 > n_0$, the prediction is *taken*, the misprediction probability is $1 - q$, and we obtain a formula for $\widehat{m}$ similar to the one above but with n_1 and n_0 permuted. Using the properties of the beta function, more precisely $B(x+1, y) = \frac{x}{x+y} B(x, y)$ and $B(x, y) = B(y, x)$, we obtain the following simple formula:

$$\widehat{m} = \frac{1 + \min(n_1, n_0)}{2 + n_1 + n_0} \tag{1}$$

Formula (1) is known as Laplace's rule of succession. Notice that $\widehat{m} = 1/2$ when $n_0 = n_1$. Table 1 gives the value of $\widehat{m}$ for n_1 and n_0 in $[0, 7]$.

Estimator $\widehat{m}$ provides an estimation of branch predictability, with low $\widehat{m}$ meaning high confidence in the prediction. Going back to the previous example, prediction A gives $\widehat{m}(2, 0) = 0.25$, prediction B gives $\widehat{m}(3, 6) \approx 0.36$. Hence prediction A is deemed more reliable than prediction B.

It must be emphasized that $\widehat{m}$ is an *estimate* of the misprediction probability, not the actual misprediction probability, which is unknown to us. Moreover, there is no reason for q to be uniformly distributed in $[0, 1]$. The actual distribution of q is an empirical property depending on the application and on the path length. In particular, $\widehat{m}$ ignores global correlations, which tend to push q away from 1/2. Nevertheless, I observed in my experiments the practical efficacy of estimator $\widehat{m}$, which is the basis of the BATAGE (Bayesian TAGE) predictor described in the following sections.

4.2 Defining confidence levels

BATAGE does not use the full spectrum of $\widehat{m}$ values, but coarsens it into a few confidence levels: high, medium and low. In this paper, medium confidence is synonymous with unknown predictability. Notice that a single occurrence of a branch gives no information about its predictability. Therefore, the case $n_1 + n_0 = 1$ is a situation where predictability is unknown. This corresponds to $\widehat{m} = 1/3$, which is the natural threshold to use here. This leads to the following confidence levels:

$$\begin{aligned} \widehat{m} < 1/3 &\longrightarrow \text{high} \\ \widehat{m} = 1/3 &\longrightarrow \text{medium} \\ \widehat{m} > 1/3 &\longrightarrow \text{low} \end{aligned}$$

Table 1 uses a color code for the confidence levels: red for high confidence, orange for medium confidence and black for low confidence.

A hardware implementation does not need to compute $\widehat{m}$ explicitly. It is sufficient to compute a value

counter width	2 bits	3 bits	4 bits	5 bits
predictor size	8KB	12KB	16KB	20KB
average MPKI	10.47	10.07	10.12	10.29

Table 2: MPKI of a 32k-entry bimodal predictor on the CBP 2016 traces.

$\overline{m} \in \{0, 1, 2\}$ directly from n_1 and n_0:

$$\begin{array}{rcl} \overline{m} = 0 & \longrightarrow & \text{high} \\ \overline{m} = 1 & \longrightarrow & \text{medium} \\ \overline{m} = 2 & \longrightarrow & \text{low} \end{array}$$

The 2-bit value $\overline{m}$ is obtained with simple combinational logic as follows. Define

$$\begin{array}{rcl} medium & = & (n_1 = (2n_0 + 1)) \quad \vee \quad (n_0 = (2n_1 + 1)) \\ low & = & (n_1 < (2n_0 + 1)) \quad \wedge \quad (n_0 < (2n_1 + 1)) \end{array}$$

then $\overline{m}$ is the concatenation of bits low and $medium$:

$$\overline{m} = 2 \times low + medium$$

5 BATAGE

Like TAGE, BATAGE does not name one particular predictor configuration but a predictor family based on some common principles. This section describes those principles, focusing on what distinguishes BATAGE from TAGE. The parts of the predictor that are not described are essentially similar to TAGE.

5.1 The dual-counter

So far, we have assumed that n_1 and n_0 can increase indefinitely. In practice however, the n_1 and n_0 counters in a BATAGE entry are narrow (typically, 3 bits each), for the same reasons that TAGE has narrow prediction counters: under limited storage, the statistical benefit of using wide counters is not high enough to justify using storage for this, and using wide counters actually hurts prediction accuracy for certain branches not behaving like a Bernoulli process. For instance, Table 2 gives the average MPKI of a large bimodal predictor for different widths of the up/down counter. On the CBP 2016 traces, a large bimodal predictor is optimal with 3-bit up/down counters. 5-bit counters are significantly worse than 3-bit counters because of non-Bernoulli branch behaviors.

So we must deal with the fact that n_1 and n_0 cannot exceed a limit $nmax$ (with 3-bit counters, $nmax = 7$). BATAGE uses the following update method (C language):

```
if (taken) { // branch is taken
  if (n1 < nmax) n1++;
  else if (n0>0) n0--;
} else { // branch is not taken
  if (n0 < nmax) n0++;
  else if (n1>0) n1--;
}
```

In other words, the counter corresponding to the branch direction is incremented, unless that counter equals $nmax$, in which case the other counter is decremented if not null. I call (n_1, n_0) and the algorithm above the *dual-counter* automaton.

It should be noted that, after having hit the limit $nmax$ once, the dual-counter behaves like an up/down counter with $2 \times nmax + 1$ states. Although formula (1) was established assuming unlimited n_0 and n_1, BATAGE always uses the confidence levels shown in Table 1.

The dual-counter is used only in tagged entries, not in the tagless table, where this would be a waste of storage. The tagless table of BATAGE contains conventional up/down counters, like TAGE.

5.2 BATAGE prediction algorithm

Let $L_1, L_2, \cdots, L_G$ denote the path lengths from shortest to longest, where G is the number of predefined path lengths. Similar to TAGE, a BATAGE prediction starts by reading simultaneously the tagless entry and the G tagged entries.

BATAGE computes the 2-bit confidence level $\overline{m}_i$ for all tagged entries $i \in [1, G]$, as described in Section 4.2. This computation is done in parallel with tag comparisons. When there is a tag miss at L_i, we set $\overline{m}_i = 3$.

We also define a confidence level $\overline{m}_0 \in \{0, 1, 2\}$ for the tagless entry, as follows. The tagless entry of BATAGE contains a 3-bit up/down counter $x \in [-4, 3]$. The value $\overline{m}_0$ is set according to the following equivalence:

x	-4	-3	-2	-1	0	1	2	3
n_1	1	1	1	1	2	3	4	5
n_0	5	4	3	2	1	1	1	1
$\overline{m}_0$	0	0	1	2	2	1	0	0

The BATAGE prediction algorithm is (C-like pseudo-code):

```
j = G
for (i=G-1; i>=0; i--) {
  if (m̄_i < m̄_j) j=i;
}
// entry j provides the final prediction
```

In other words, the final prediction is provided by the entry with the smallest $\overline{m}$ value, with priority to the longest path in case of equality. The prediction selection circuit of BATAGE is no more complex than that of TAGE. A divide-and-conquer approach is possible, allowing a tree structure, as illustrated in Figure 5. Other implementations are possible. In some implementations, it might be advantageous to use a 3-bit thermometer code for $\overline{m}$ ($0 \rightarrow 000$, $1 \rightarrow 001$, $2 \rightarrow 011$, $3 \rightarrow 111$), which allows to compute the minimum of multiple $\overline{m}$ values with a bitwise AND.

5.3 Decay mechanism

In TAGE, the u counter is for protecting useful entries against eviction. There are no u counters in BATAGE. Instead, we use the confidence level as an indication of utility: a high confidence prediction is more likely to be useful than a medium or low confidence one. More precisely, the following rule is used in BATAGE:

a tagged entry giving a high confidence prediction cannot be evicted.

Yet, a high confidence entry that is no longer used should not be locked forever. For such entry to become evictable, there must be a way to *decay* it. We define the action of decaying an entry as follows (C language) :

```
if (n1 > n0) n1--;
if (n0 > n1) n0--;
```

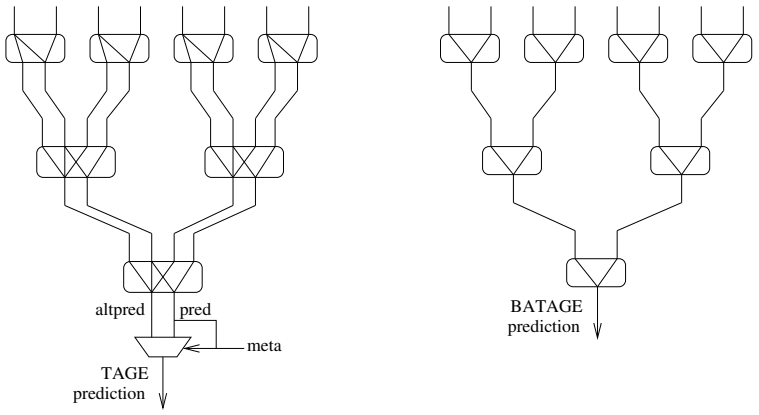

Figure 5: Divide-and-conquer implementation of the prediction selection circuit for TAGE (left) and BATAGE (right). The circuit for TAGE takes as input the hit/miss bit, the prediction bit, and the low-confidence bit (needed for the final selection between *pred* and *altpred*). The circuit for BATAGE takes as input the $\overline{m}$ value (2 bits) and the prediction bit. This example assumes 7 tagged banks. The path lengths increase from left to right. The leftmost input is the bimodal entry.

entry	action	condition	comment
$L_h > L_p$	update with branch direction	always	entries were skipped probably because they are cold
L_p	update with branch direction	$L_p = L_0$ or L_p not high confidence or L_n not high confidence or incorrect prediction from L_n	
	decay	$L_p > L_0$ and L_p high confidence and L_n high confidence and correct prediction from L_n	entry is probably useless
L_n	update with branch direction	$L_p > L_0$ and L_p not high confidence	improves prediction accuracy only slightly

Table 3: Updating the hitting entries. L_0 is the tagless entry. L_p is the path length that provided the BATAGE prediction. Hitting entries are denoted L_h (L_0 is one of them). When $L_p > L_0$, L_n denotes the longest L_h such that $L_h < L_p$.

In other words, we decay an entry by decrementing its highest count, n_1 or n_0. Decay leaves the numerator in formula (1) unchanged but decreases the denominator. Hence decay increases $\widehat{m}$. In particular, decaying a high confidence dual-counter multiple times eventually leads to a medium confidence state (see Table 1). In practice, it is unnecessary to decay a dual-counter further than medium confidence.

5.4 Updating the hitting entries

Predictor update is done at the in-order retirement pipeline stage [63]. We define $L_1, L_2, \cdots, L_G$ as in Section 5.2. Moreover, let L_0 denote the tagless entry. At update, BATAGE needs the following information: which path length L_p provided the BATAGE prediction (Section 5.2), and all the path lengths L_h corresponding to a tag hit (including L_0). Table 3 describes how the hitting entries are updated. When $L_p > L_0$, L_n denotes the longest L_h such that $L_h < L_p$. That is, L_n is the next hitting entry after the entry that provided the BATAGE prediction. Entries $L_h < L_n$ are not updated. Entries $L_h > L_p$ are always updated with

the branch direction, as described in Section 5.1. Those entries exist because shorter path lengths could not deliver 100% accuracy, but they were skipped probably because they are cold. By being updated, they can warm up and become useful. The dual-counter in entry L_p is decayed when that entry is deemed useless, that is, when $L_p > L_0$ and the prediction from L_n is correct and L_p and L_n are both high confidence.

5.5 Allocating new entries

TAGE-like predictors allocate tagged entries on branch mispredictions because it is hoped that longer paths will provide more accurate predictions. Allocating a tagged entry means setting the tag and initializing the dual-counter according to the branch outcome ($n_0 + n_1 = 1$). Like TAGE, BATAGE allocates entries only upon mispredictions, at retirement. BATAGE allocates at most one entry per misprediction, like the original 2006 TAGE.

Let L_m be the longest hitting path length ($L_m = L_0$ when there are no hits), and $s \geq 1$ a small random number. The allocation algorithm selects the entry corresponding to the shortest path length $L_r \geq L_{m+s}$ such that the entry at L_r is not high confidence ($\overline{m} \neq 0$). If such L_r does not exist, no allocation is done, and we set $r = G + 1$. Then, independent of allocation's success, the entries $L_i \in [L_{m+s}, L_{r-1}]$ (which are all high confidence) are decayed with probability PDEC, as described in Section 5.3. The best value of PDEC is determined empirically. PDEC=$1/4$ works well on the CBP 2016 traces.

5.6 Controlled allocation throttling (CAT)

While Bayesian confidence estimation helps mitigate the cold-counter problem, it does not solve the root cause, which is the allocation of tagged entries on every branch misprediction.

I observed that, on traces with hard-to-predict branches, prediction accuracy improves when the allocation probability is set to a value less than one, which I call *allocation throttling*. This observation is valid both for TAGE and BATAGE. The problem is that allocation throttling degrades prediction accuracy severely on some traces. In fact, an allocation probability of one, as in TAGE, is the best fixed allocation probability on average.

The problem is to find the best allocation probability automatically. That is, we are looking for an effective *controlled allocation throttling* (CAT) method. After trying several methods that did not work, I made progress toward a solution when I conjectured that one could tell whether there are too many or too few allocations by looking at the global predictor state, more precisely the distribution of dual-counter states. I noticed that, when entries are allocated too aggressively, many entries are in medium or low confidence state, either because allocated entries are not reused, or because these entries are useless and kept in medium/low confidence state by the update rules (Table 3). Conversely, medium/low confidence entries are very few when the allocated entries are quickly reused and converted into high confidence entries. However, my attempts to adjust allocation throttling to obtain a fixed number of medium/low confidence entries did not produce good results. Then, I realized that what matters is not the absolute number of medium/low confidence entries but their proportion with respect to the number of *moderately high* confidence entries. The rest of this section provides a detailed description of the CAT method I found.

The main idea is to compare the amount of entries in a medium/low confidence state, denoted **NHC** (for *not high confidence*), and the amount of entries in a *moderately high confidence* state, denoted **MHC**, where MHC states are the states such that $0.17 < \widehat{m} < 1/3$. That is, the MHC states are the high confidence states from which are excluded $n_0 = 0$, $n_1 \geq 4$ and $n_1 = 0$, $n_0 \geq 4$ (see Table 1). MHC states are generally transient states. Besides statistical fluctuations due to random branch behavior, allocation and decay are two major causes for the existence of MHC entries.

```
if (misprediction) {
  r = random in [0,MINAP-1];
  if (r >= ((cat*MINAP)/(CATMAX+1))) {
    mhc = 0;
    s = random in [1,SKIPMAX];
    for (i=m+s; i ≤ G; i++) {
      if (m̂[i] < 1/3) { // high confidence, protected
        decay with probability PDEC;
        if (m̂[i] > 0.17) mhc++;
      } else { // not high confidence, take the entry
        overwrite tag;
        initialize dual-counter;
        cat = cat + 3 - 4 * mhc; // CATR=3/4
        cat = min(CATMAX, max(0, cat));
        break; // stop here
      }
    }
  }
}
```

Figure 6: BATAGE allocation algorithm (C-like pseudo-code). L_m is the longest hitting path length, $\widehat{m}[i]$ is the $\widehat{m}$ estimator for the entry at L_i. The cat counter is initially set to zero.

When NHC entries significantly outnumber MHC entries, this is generally because allocation is easy (little decay) but allocated entries stay in a medium or low confidence state, hence allocation is probably too aggressive. Conversely, when MHC entries outnumber NHC entries, this is an indication that allocation is difficult (much decay) and allocated entries quickly evolve toward a high confidence state, so allocation can be more aggressive.

A possible implementation of CAT would be to monitor the fractions F_n and F_m of NHC and MHC entries respectively. In practice though, we just want to know the ratio F_m/F_n. This ratio can be estimated as follows. Upon an allocation attempt, the average number of skipped entries is $1/F_n - 1$, neglecting the fact that the number G of tagged banks is finite. The fraction of high confidence entries that are MHC is $F_m/(1 - F_n)$. Hence the average number of skipped MHC entries is

$$\left(\frac{1}{F_n} - 1\right) \times \frac{F_m}{1 - F_n} = \frac{F_m}{F_n}$$

which is the quantity we are looking for.

When the average number of skipped MHC entries is below a certain threshold CATR, the allocation probability is decreased if possible. The allocation probability cannot fall below a value 1/MINAP. When the average number of skipped MHC entries is greater than CATR, the allocation probability is increased if possible. Typically, the optimal CATR value lies between 1/4 and 3/4 and it tends to increase with G.

Figure 6 describes the BATAGE allocation algorithm. The cat counter controls the allocation probability. The allocation probability is 1 when cat is small, it is 1/MINAP when cat is close to CATMAX. The maximum cat value, CATMAX, is typically much larger than the total number of tagged entries. The predictor can warm up more quickly if cat is reset to zero when an application starts running.

6 Experimental evaluation

I evaluated BATAGE with the CBP 2016 simulation infrastructure, which provides a set of 220 traces for tuning predictors and a distinct set of 440 traces for evaluating them [6].

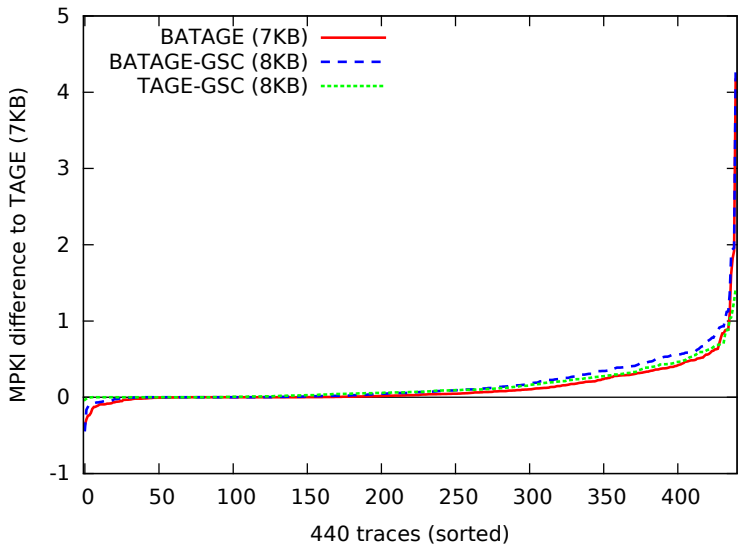

Figure 7: MPKI difference to 7KB TAGE for BATAGE, TAGE-GSC and BATAGE-GSC (higher is better). The traces are sorted for each curve separately so that the curve is non-decreasing.

The average MPKIs of TAGE and BATAGE are shown in Figure 1 (2016 and 2017) for 8KB, 32KB and 64KB storage budgets. To obtain a stand-alone 8KB TAGE, I started from the 7KB TAGE embedded in the 8KB TAGE-SC-L that won CBP 2016 [66], and I tuned it without the statistical corrector. In particular, I replaced 3-bit counters in tagged entries with 4-bit counters (this improves the average prediction accuracy) and I scaled the predictor up to 8KB by increasing the number of banks and the number of path lengths. For the 32KB and 64KB TAGE, I proceeded similarly but starting from the 56KB TAGE embedded in the winning 64KB TAGE-SC-L.

The prediction accuracy gain of BATAGE over TAGE is modest on average, but tangible: it is roughly equivalent to the accuracy gain that the 2001 perceptron brought over an e-gskew. The 8KB BATAGE even matches the MPKI of the CBP 2016 winner, despite being simpler (no statistical corrector, no local history, no loop predictor).

The rest of this section provides a detailed evaluation of BATAGE, focusing on a 7KB storage budget. The 7KB TAGE component embedded in the 8KB TAGE-SC-L of CBP 2016 features two u bits per tagged entry. After replacing the 3-bit up/down counter in the tagged entry of TAGE with a 4-bit counter, the tagged entries of TAGE and BATAGE have the same payload size (6 bits). This way, the 7KB TAGE and BATAGE configurations can be compared with equal tag size and equal number of entries and banks.

6.1 BATAGE does not need statistical correction

Figure 7 and Table 4 compare TAGE and BATAGE with and without a global-history statistical corrector (GSC). The GSC is the 1KB statistical corrector implemented in the 8KB TAGE-SC-L of CBP 2016, but with the local-history components disabled so that the effect is pure statistical correction. Figure 7 shows the MPKI difference to the 7KB TAGE (higher is better) for the 440 traces. Table 4 provides the average MPKI (arithmetic mean over the 440 traces).

BATAGE outperforms TAGE significantly on several tens of traces (rightmost part of the red curve). The statistical corrector helps TAGE significantly, as expected [63]. The average MPKI of the 8KB TAGE-GSC

predictor	TAGE	TAGE-GSC	BATAGE	BATAGE-GSC
size	7KB	8KB	7KB	8KB
MPKI	4.42	4.27	4.29	4.24

Table 4: Average MPKI of TAGE and BATAGE with and without statistical correction.

CAT	enabled			disabled		
prediction selection	$k = G + 1$	$k = 2$	$k = 1$	$k = G + 1$	$k = 2$	$k = 1$
MPKI	4.29	4.38	4.62	4.43	4.58	4.77

Table 5: BATAGE: impact of disabling CAT and restricting the prediction algorithm.

is very close to that of the 7KB BATAGE. However, the 8KB BATAGE-GSC is only marginally better than the 7KB BATAGE. In practice, BATAGE does not need statistical correction.

6.2 Controlled allocation throttling is an essential feature of BATAGE

Two features of BATAGE contribute to solve the cold-counter problem:

- Bayesian confidence estimation takes into account the "temperature" of tagged entries,
- Controlled allocation throttling decreases the number of allocations hence the number of cold entries.

The impact of the BATAGE prediction algorithm can be evaluated by restricting the selection of the final prediction among the first k hitting entries. The case $k = G + 1$ is the BATAGE prediction algorithm. The case $k = 1$ corresponds to selecting the prediction from the longest hitting path, as in the PPM-like predictor [45]. The case $k = 2$ corresponds to selecting between the longest and second longest hitting paths, as in TAGE (but without a meta-predictor).

The impact of CAT can be evaluated by disabling it, that is, by setting the allocation probability equal to one. When CAT is disabled and k is small, the use of decay in the update policy (Table 3) hurts prediction accuracy because the safety net provided by the BATAGE prediction algorithm has been removed. In these cases (no CAT, small k), the update policy must be modified so that the entry providing the final prediction is always updated.[7]

Table 5 gives the average MPKI for the 7KB BATAGE and for configurations with CAT disabled and/or a small k. Both disabling CAT and limiting k hurt prediction accuracy. Figure 8 shows, for each trace, the MPKI degradation from disabling CAT. Several tens of traces benefit substantially from CAT (leftmost part of the curve), and a few traces experience a degradation, yet barely noticeable (rightmost part of the curve).

CAT is an essential feature of BATAGE: without it, BATAGE would be no better than TAGE (compare tables 4 and 5). The management of u counters in TAGE is very effective at preserving useful entries. BATAGE has no u counter and instead relies on dual-counter decay. However, without CAT, dual-counters are decayed too aggressively on certain traces. CAT is effective not only at reducing the number of allocations but also at adjusting decay intensity.

Figure 9 compares the number of allocations per 1000 instructions (APKI) of TAGE and BATAGE, showing also the impact of disabling CAT. Without CAT, BATAGE would do more allocations than TAGE on average. This is because TAGE does not allocate entries when the overall prediction is incorrect but

[7] However the use of decay is important for CAT, as it is one of the mechanisms allowing to detect useless allocations by keeping useless entries in a medium confidence state.

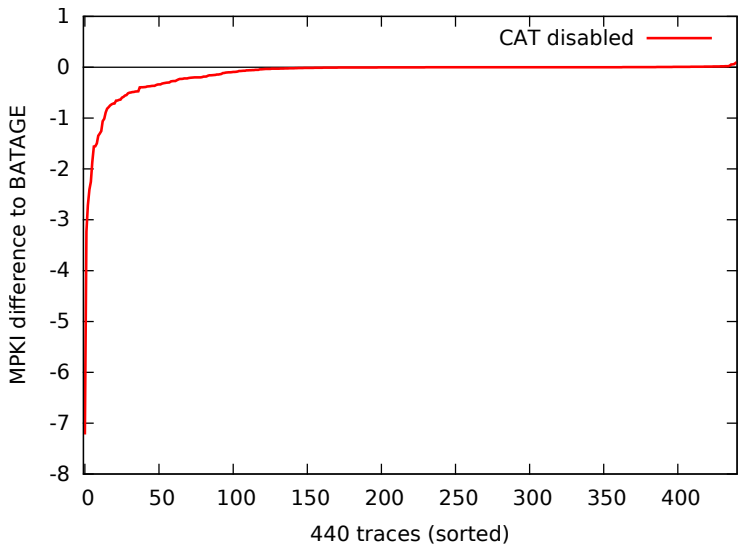

Figure 8: Impact of disabling controlled allocation throttling on a 7KB BATAGE.

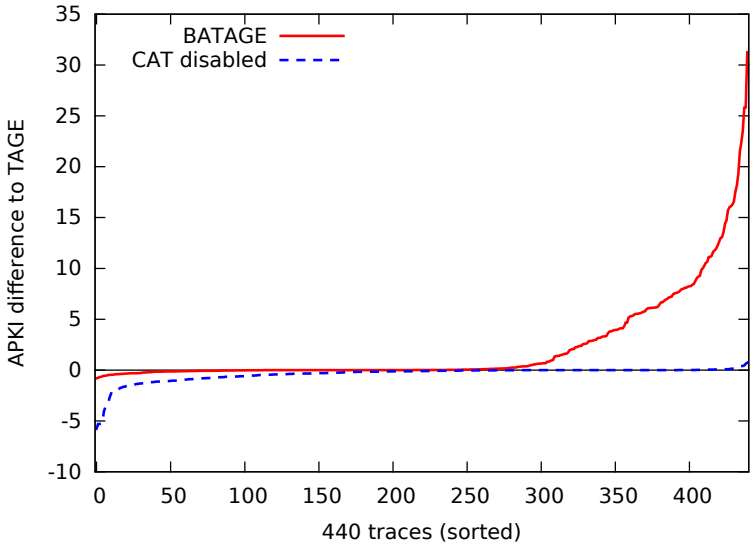

Figure 9: APKI difference to TAGE (higher means fewer allocations).

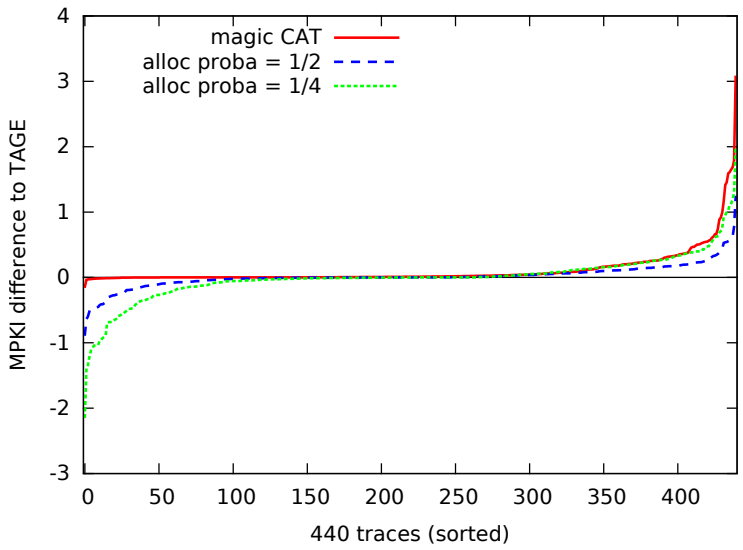

Figure 10: Allocation throttling on TAGE. The dashed curves are for a fixed allocation probability. The curve labeled "magic CAT" is for TAGE with a CAT counter provided by BATAGE.

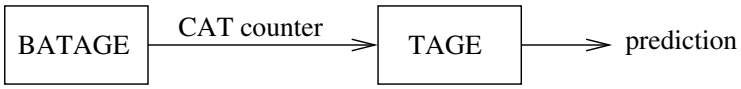

Figure 11: Doing CAT "magically" on TAGE.

the prediction from the longest matching path is correct.[8] However, with CAT enabled, the number of allocations is reduced dramatically on many traces (Figure 9, right part of the red curve).

6.3 CAT on TAGE?

Given the benefit brought by CAT on BATAGE, it would be satisfying to find a CAT method for TAGE.[9] The dashed curves in Figure 10 show the prediction accuracy change when the allocation probability in TAGE is set to a fixed value, here $1/2$ and $1/4$. Allocation throttling with a fixed allocation probability improves prediction accuracy on certain traces but degrades it on other traces. In TAGE as in BATAGE, allocation throttling must be controlled. However, the CAT method described in Section 5.6 relies on specificities of BATAGE. It is not clear how TAGE could imitate BATAGE. This is a question for future research.

Nevertheless, it is possible to evaluate the potential prediction accuracy improvements that a hypothetical CAT method would bring to TAGE. Figure 11 depicts the experiment I did. I simulated a BATAGE in lockstep with TAGE, the cat counter of BATAGE being communicated to TAGE continuously so that TAGE can set its allocation probability like BATAGE at every instant. The result of this experiment is shown in Figure 10 as the solid red curve. The rightmost part of the curve shows that there is a potential for significant improvement on several tens of traces, motivating future research on CAT for TAGE.

[8] This feature of TAGE decreases the number of allocations and brings modest MPKI reductions.

[9] In a private discussion I had with André Seznec in july 2017, he told me that he was aware of the potential benefit of allocation throttling but had not found a way to control it.

7 Conclusion

The dual-counter is central to BATAGE. It permits solving the cold-counter problem by allowing Bayesian confidence estimation and by providing a way to control allocation throttling. My original motivation for getting rid of the u counter of TAGE was that the dual-counter needs more bits than a conventional up/down counter, and keeping the u counter would have made the tagged entry larger. Although dual-counter decay allows to recover some of the benefit of the u counter, the two mechanisms work differently, and perhaps they provide distinct advantages. Maybe future advances on TAGE-like prediction will come from a better understanding of the differences between TAGE and BATAGE. Nevertheless, dual-counter decay seems to be essential for CAT to work on BATAGE. Finding a CAT method for TAGE is a topic for future research.

A Legend for Figure 1

Figure 1 shows the average MPKI (arithmetic mean over the CBP 2016 traces) of various conditional branch predictors over the years, at 8KB, 32KB and 64KB storage budgets. The predictors are *gshare* [42], the 1993 *bimodal/gshare* hybrid [42], a *bimodal/gshare* where the meta-prediction table is accessed by hashing together the branch address and the global branch history [44, 10], *e-gskew* [47], *bimode* [39], YAGS [17], the original *2bc-gskew* [68], a *2bc-gskew* tuned along the lines of the Alpha EV8 predictor [67, 55], the original global-history *perceptron* [35], PBNP, [31], MAC-RHSP [58], PWL [32], O-GEHL [59], the PPM-like predictor [45], TAGE [69], L-TAGE [61], PMPM [22], FTL [27], ISL-TAGE [62, 63], FTL++ [28], OH-SNAP [33, 73], TAGE-SC-L [65, 66], SSHP [34], BFN [24, 23], MPP [29], and BATAGE (this paper).

All the predictors were evaluated on the 440 CBP 2016 traces [6]. Predictors whose name is followed by the mention "CBP" in parentheses participated in a branch prediction championship. For these predictors, I used the original source code written by the authors, downloaded from the championship's web site. I did not modify the source code except for the minimal modifications necessary to simulate the predictor under the CBP 2016 simulation infrastructure.[10] I could not simulate the winning predictor of CBP 2004 [21], as it uses information not available in the CBP 2016 traces. Also, some CBP predictors are omitted for clarity.

For the 2bc-gskew-EV8, MAC-RHSP and TAGE 2006 predictors, I used the source code downloaded from André Seznec's website [54]. I obtained the TAGE 2016 predictor by disabling the statistical corrector in Seznec's CBP 2016 source code and by tuning TAGE parameters to fit the storage budget considered. I implemented the other predictors from the descriptions provided in the original papers.

I tuned the non-CBP predictors using the 223 training traces of CBP 2016. This tuning was done by varying the explicit parameters (for instance, the path length). I did not modify the original algorithms except for the bimode and YAGS predictors, which I made slightly more accurate with algorithm modifications simple enough to be considered part of the tuning.

All the non-CBP predictors and five CBP predictors (PPM-like, O-GEHL, L-TAGE, ISL-TAGE, BFN) use only a global branch or path history as first history level. The predictors using the perceptron algorithm are shown in red, those not using it in blue (the statistical corrector in ISL-TAGE and TAGE-SC-L is GEHL-like).

[10] The CBP 2016 traces are based on the ARM architecture, while previous championships were based on the Intel x86 architecture. Hence, for the old CBP predictors, I right-shifted every branch address by 2 bits.

References

[1] J. Bonnano, A. Collura, D. Lipetz, U. Mayer, B. Prasky, and A. Saporito. Two level bulk preload branch prediction. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

[2] CBP. Championship branch prediction (CBP-1), December 2004. `https://www.jilp.org/cbp/`.

[3] CBP. Championship branch prediction (CBP-2), December 2006. `http://hpca23.cse.tamu.edu/taco/camino/cbp2/`.

[4] CBP. Championship branch prediction (CBP-3), June 2011. `https://www.jilp.org/jwac-2/`.

[5] CBP. Championship branch prediction (CBP-4), June 2014. `https://www.jilp.org/cbp2014/`.

[6] CBP. Championship branch prediction (CBP-5), June 2016. `https://www.jilp.org/cbp2016/`.

[7] P.-Y. Chang, M. Evers, and Y. N. Patt. Improving branch prediction accuracy by reducing pattern history table interference. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 1996.

[8] P.-Y. Chang, E. Hao, and Y. N. Patt. Target prediction for indirect jumps. In *International Symposium on Computer Architecture (ISCA)*, 1997.

[9] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. Patt. Branch classification: a new mechanism for improving branch predictor performance. In *International Symposium on Microarchitecture (MICRO)*, 1994.

[10] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. N. Patt. Alternative implementations of hybrid branch predictors. In *International Symposium on Microarchitecture (MICRO)*, 1995.

[11] I-C. Chen. *Enhancing the instruction fetching mechanism using data compresssion*. PhD thesis, University of Michigan, 1997.

[12] I-C. K. Chen, J.T. Coffey, and T.N. Mudge. Analysis of branch prediction via data compression. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1996.

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

[14] K. Driesen and U. Hölzle. Accurate indirect branch prediction. In *International Symposium on Computer Architecture (ISCA)*, 1998.

[15] K. Driesen and U. Hölzle. The cascaded predictor: economical and adaptive branch target prediction. In *International Symposium on Microarchitecture (MICRO)*, 1998.

[16] K. Driesen and U. Hölzle. Multi-stage cascaded prediction. In *International Euro-Par Conference on Parallel Processing*, 1999. LNCS 1685.

[17] A. N. Eden and T. Mudge. The YAGS branch prediction scheme. In *International Symposium on Microarchitecture (MICRO)*, 1998.

[18] M. Evers. *Improving branch prediction by understanding branch behavior*. PhD thesis, University of Michigan, 2000.

[19] M. Evers, P.-Y. Chang, and Y. N. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In *International Symposium on Computer Architecture (ISCA)*, 1996.

[20] E. Federovsky, M. Feder, and S. Weiss. Branch prediction based on universal data compression algorithms. In *International Symposium on Computer Architecture (ISCA)*, 1998.

[21] H. Gao and H. Zhou. Adaptive information processing: an effective way to improve perceptron predictors. *Journal of Instruction-Level Parallelism*, 7, April 2005. Special issue CBP 2004.

[22] H. Gao and H. Zhou. PMPM: prediction by combining multiple partial matches. *Journal of Instruction-Level Parallelism*, 9, May 2007. Special issue CBP 2006.

[23] D. Gope and M. H. Lipasti. Bias-free branch predictor. In *International Symposium on Microarchitecture (MICRO)*, 2014.

[24] D. Gope and M. H. Lipasti. Bias-free neural predictor. In *Championship Branch Prediction (CBP-4)*, 2014. https://www.jilp.org/cbp2014/.

[25] R. N. Ibbett. The MU5 instruction pipeline. *The Computer Journal*, 15(1), February 1972.

[26] IBM. POWER9 processor user's manual. Version 2.0, April 2018. https://openpowerfoundation.org.

[27] Y. Ishii. Fused two-level branch prediction with ahead calculation. *Journal of Instruction-Level Parallelism*, 9, May 2007. Special issue CBP 2006.

[28] Y. Ishii, K. Kuroyanagi, T. Sawada, M. Inaba, and K. Hiraki. Revisiting local history for improving fused two-level branch prediction. In *Championship Branch Prediction (CBP-3)*, 2011. https://www.jilp.org/jwac-2/.

[29] D. Jiménez. Multiperspective perceptron predictor. In *Championship Branch Prediction (CBP-5)*, 2016. https://www.jilp.org/cbp2016/.

[30] D. Jiménez. Multiperspective perceptron predictor with TAGE. In *Championship Branch Prediction (CBP-5)*, 2016. https://www.jilp.org/cbp2016/.

[31] D. A. Jiménez. Fast path-based neural branch prediction. In *International Symposium on Microarchitecture (MICRO)*, 2003.

[32] D. A. Jiménez. Piecewise linear branch prediction. In *International Symposium on Computer Architecture (ISCA)*, 2005.

[33] D. A. Jiménez. OH-SNAP: optimized hybrid scaled neural analog predictor. In *Championship Branch Prediction (CBP-3)*, 2011. https://www.jilp.org/jwac-2/.

[34] D. A. Jiménez. Strided sampling hashed perceptron predictor. In *Championship Branch Prediction (CBP-4)*, 2014. https://www.jilp.org/cbp2014/.

[35] D. A. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2001.

[36] D. A. Jiménez and C. Lin. Composite confidence estimators for enhanced speculation control. Technical Report TR-02-14, The University of Texas at Austin, Department of Computer Sciences, January 2002.

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

[38] J. Kalamatianos and D. R. Kaeli. Predicting indirect branches via data compression. In *International Symposium on Microarchitecture (MICRO)*, 1998.

[39] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge. The bi-mode branch predictor. In *International Symposium on Microarchitecture (MICRO)*, 1997.

[40] J. K. F. Lee and A. J. Smith. Branch prediction strategies and branch target buffer design. *Computer*, 17(1), January 1984.

[41] G. H. Loh and D. S. Henry. Predicting conditional branches with fusion-based hybrid predictors. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2002.

[42] S. McFarling. Combining branch predictors. Technical Report TN-36, DEC WRL, 1993.

[43] S. McFarling. Branch predictor with serially connected predictor stages for improving branch prediction accuracy. US patent 6374349, March 1998.

[44] S. McFarling, S. C. Steely Jr., J. Emer, and E. McLellan. Trainable apparatus for predicting instruction outcomes in pipelined processors. US patent 5758142, May 1994.

[45] P. Michaud. A PPM-like, tag-based predictor. *Journal of Instruction-Level Parallelism*, 7, April 2005. Special issue CBP 2004.

[46] P. Michaud and A. Seznec. A comprehensive study of dynamic global history branch prediction. Technical Report RR-4219, INRIA, June 2001.

[47] P. Michaud, A. Seznec, and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *International Symposium on Computer Architecture (ISCA)*, 1997.

[48] R. Nair. Dynamic path-based branch correlation. In *International Symposium on Microarchitecture (MICRO)*, 1995.

[49] R. Nair. Optimal 2-bit branch predictors. *IEEE Transactions on Computers*, 44(5), May 1995.

[50] S.-T. Pan, K. So, and J. T. Rameh. Improving the accuracy of dynamic branch prediction using branch correlation. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1992.

[51] C. G. Ponder and M. C. Shebanow. An information-theoretic look at branch-prediction. In "Studies in branch-prediction" by C.G. Ponder. UCRL-ID-106077, Lawrence Livermore National Laboratory, September 1990.

[52] D. J. Schlais and M. H. Lipasti. BADGR: a practical GHR implementation for TAGE branch predictors. In *International Conference on Computer Design (ICCD)*, 2016.

[53] S. Sechrest, C.-C. Lee, and T. Mudge. Correlation and aliasing in dynamic branch predictors. In *International Symposium on Computer Architecture (ISCA)*, 1996.

[54] A. Seznec. `https://team.inria.fr/pacap/members/andre-seznec/`.

[55] A. Seznec. An optimized 2bcgskew branch predictor. `https://team.inria.fr/pacap/members/andre-seznec/`, September 2003.

[56] A. Seznec. Redundant history skewed perceptron predictors: pushing limits on global history branch predictors. Technical Report PI-1554, IRISA, September 2003.

[57] A. Seznec. The O-GEHL branch predictor. In *Championship Branch Prediction (CBP-1)*, December 2004.

[58] A. Seznec. Revisiting the perceptron predictor. Technical Report PI-1620, IRISA, May 2004.

[59] A. Seznec. Analysis of the O-geometric history length branch predictor. In *International Symposium on Computer Architecture (ISCA)*, 2005.

[60] A. Seznec. The idealistic GTL predictor. *Journal of Instruction-Level Parallelism*, 9, May 2007. Special issue CBP 2006.

[61] A. Seznec. The L-TAGE branch predictor. *Journal of Instruction-Level Parallelism*, 9, May 2007. Special issue CBP 2006.

[62] A. Seznec. A 64 Kbytes ISL-TAGE branch predictor. In *Championship Branch Prediction (CBP-3)*, 2011. `https://www.jilp.org/jwac-2/`.

[63] A. Seznec. A new case for the TAGE branch predictor. In *International Symposium on Microarchitecture (MICRO)*, 2011.

[64] A. Seznec. Storage free confidence estimation for the TAGE branch predictor. In *International Symposium on High Performance Computer Architecture (HPCA)*, 2011.

[65] A. Seznec. TAGE-SC-L branch predictors. In *Championship Branch Prediction (CBP-4)*, 2014. `https://www.jilp.org/cbp2014/`.

[66] A. Seznec. TAGE-SC-L branch predictors again. In *Championship Branch Prediction (CBP-5)*, 2016. https://www.jilp.org/cbp2016/.

[67] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides. Design tradeoffs for the Alpha EV8 conditional branch predictor. In *International Symposium on Computer Architecture (ISCA)*, 2002.

[68] A. Seznec and P. Michaud. De-aliased hybrid branch predictors. Technical Report RR-3618, INRIA, February 1999.

[69] A. Seznec and P. Michaud. A case for (partially) tagged geometric history length branch prediction. *Journal of Instruction-Level Parallelism*, 8, February 2006.

[70] J. E. Smith. A study of branch prediction strategies. In *International Symposium on Computer Architecture (ISCA)*, 1981.

[71] M. K. Smotherman, E. H. Sussenguth, and R. J. Robelen. The IBM ACS project. *IEEE Annals of the History of Computing*, 38(1), January 2016.

[72] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt. The agree predictor: a mechanism for reducing negative branch history interference. In *International Symposium on Computer Architecture (ISCA)*, 1997.

[73] R. St. Amant, D. A. Jiménez, and D. Burger. Low-power, high-performance analog neural branch prediction. In *International Symposium on Microarchitecture (MICRO)*, 2008.

[74] A. R. Talcott, M. Nemirovsky, and R. C. Wood. The influence of branch prediction table interference on branch prediction scheme performance. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 1995.

[75] D. Tarjan and K. Skadron. Revisiting the perceptron predictor again. Technical Report CS-2004-28, University of Virginia, September 2004.

[76] D. Tarjan and K. Skadron. Merging path and gshare indexing in perceptron branch prediction. *ACM Transactions on Architecture and Code Optimization*, 2(3), September 2005.

[77] L. N. Vintan and M. Iridon. Towards a high performance neural branch predictor. In *International Joint Conference on Neural Networks (IJCNN)*, 1999.

[78] T.-Y. Yeh and Y. N. Patt. Two-level adaptive training branch prediction. In *International Symposium on Microarchitecture (MICRO)*, 1991.

[79] T.-Y. Yeh and Y. N. Patt. Alternative implementations of two-level adaptive branch prediction. In *International Symposium on Computer Architecture (ISCA)*, 1992.

[80] T.-Y. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *International Symposium on Computer Architecture (ISCA)*, 1993.

[81] C. Young, N. Gloy, and M. D. Smith. A comparative analysis of schemes for correlated branch prediction. In *International Symposium on Computer Architecture (ISCA)*, 1995.

[82] C. Zhang. Mars: a 64-core ARMv8 processor. Hot Chips, 2015. http://www.hotchips.org/archives/.