A New Case for the TAGE Branch Predictor*

André Seznec
INRIA/IRISA
Campus de Beaulieu
35042 Rennes Cedex

E-mail: andre.seznec@inria.fr

Abstract

The TAGE predictor is often considered as state-of-the-art in conditional branch predictors proposed by academy. In this paper, we first present directions to reduce the hardware implementation cost of TAGE. Second we show how to further reduce the misprediction rate of TAGE through augmenting it with small side predictors.

On a hardware implementation of a conditional branch predictor, the predictor tables are updated at retire time. A retired branch normally induces three accesses to the branch predictor tables : read at prediction time, read at retire time and write for the update. We show that in practice, the TAGE predictor accuracy would not be significantly impaired by avoiding a systematic second read of the prediction tables at retire time for correct prediction. Combined with the elimination of silent updates, this significantly reduces the number of accesses to the predictor. Furthermore, we present a technique allowing to implement the TAGE predictor tables as bank-interleaved structures using single-port memory components. This significantly reduces the silicon footprint of the predictor as well as its energy consumption without significantly impairing its accuracy.

In the last few years, progress in branch prediction accuracy has relied on associating a main state-of-the-art single scheme branch predictor with specialized side predictors. As a second contribution of the paper, we develop this side predictor approach for TAGE. At the recent 3rd Championship Branch Prediction,it was shown that the TAGE predictor can be augmented with several side predictors, each one addressing a category of predictions that is not optimally addressed by TAGE. The Immediate Update Mimicker tracks the inflight already executed but not retired branches and use their result for correcting the predictions. Sometimes TAGE fails to predict when the control flow path inside the loop body is irregular. TAGE can be augmented with a loop predictor to predict loops with constant iteration numbers, TAGE also sometimes fails to predict branches that are not strongly biased but that are only statistically biased. The Statistical Corrector Predictor is introduced to track these statistically correlated branches. These side predictors allow to increase the prediction accuracy of TAGE. Building on top of these proposals, we further show that the Statistical Corrector Predictor can be used to leverage local history to further improve the potential of the TAGE predictor. Furthermore, we show that the use of a Statistical Corrector predictor based on local history, LSC, dwarfs the benefits of the loop predictor and the global history Statistical Corrector.

Finally, we present a cost-effective implementation of the TAGE-LSC predictor using single-port memory components and a reduced number of accesses per prediction.

*This work was partially supported by the European Research Council Advanced Grant DAL No 267175

Categories and Subject Descriptors

C.1.1 [**Processor Architectures**]: Single Data Stream Architectures

General Terms

Performance

1. Introduction

From 1990 to 2000, there has been a very active research in branch prediction [28, 20, 16, 4, 17, 3] resulting in the design of complex branch predictors [22]. After 2001, the research on branch prediction was essentially conducted by a small community that participated to the Championship Branch Prediction (CBP) in 2004 and 2006 [11, 21, 23, 8, 15, 6, 18]. However, tremendous progress were made during this period: the research of neural inspired predictors led by Jimenez [13, 9, 11, 1], the introduction of GEometric History Length predictors by Seznec [21] and the introduction of the TAGE predictor by Seznec and Michaud [26]. In practice, on the traces distributed for the first two CBPs, more accuracy progress was achieved between 2001 and 2006 than between 1993 and 2001 after the introduction of *gshare* by McFarling in 1993 [16].

Since 2006, TAGE has been often considered as state-of-the-art in terms of branch prediction accuracy [1, 5]. In this paper, we reenforce the case for the TAGE predictor in two directions. First we show that the TAGE predictor requires less accesses to the predictor tables than other branch predictors thus potentially enabling simpler (or more cost effective) implementation. Second, building on the side predictors recently presented at the 3rd Championship Branch Prediction [24] for the TAGE predictor, we show that TAGE can be combined with a small local history predictor to obtain even higher accuracy at the same storage budget, thus achieving state-of-art prediction accuracy. Combining these two approaches, we present a cost-effective predictor that achieves state-of-the-art prediction accuracy.

In a processor, the hardware predictor tables are updated at retire time to avoid pollution by the wrong path; this delayed update induces extra mispredictions compared with oracle update at fetch time. Previous studies were generally showing that the accuracy loss due to this delayed update on global history predictors is relatively marginal [10, 21]. However these studies are assuming that the predictor tables are read a second time and the prediction recomputed at retire time. Each branch on the correct path potentially leads to 3 accesses of the predictor tables, read at prediction time, read at retire time and write at retire time. Such a high number of accesses potentially leads to a very costly branch prediction structure (multiported and/or very complex bank-interleaved structure). As a first contribution of the paper, we show that on the TAGE predictor and contrary to other predictors, the read at retire time can be avoided with limited accuracy loss. Combined with the elimination of silent updates (more than 90 % in average), this significantly reduces the number of accesses to the predictor. Moreover, we present a bank-interleaving technique that allows to use single-ported memory components to build the TAGE predictor tables. Combining these two techniques allows significant silicon area savings and significant reduction of the energy consumption on the predictor.

Since 2006, the trend for further prediction accuracy improvement is to associate side predictors targeting special categories of branches with a state-of-the-art main predictor (TAGE, [26], OGEHL [21], Piecewise Linear [11], FTL [8]), e.g. a loop predictor with the TAGE predictor in L-TAGE [26] or the address-branch correlator in [5].

In this paper, we illustrate this approach for the TAGE predictor with the ISL-TAGE predictor, that was presented at the 3rd Championship Branch Prediction [24].

For the TAGE predictor, in order to limit the mispredictions due to delayed updates, we introduce the Immediate Update Mimicker, IUM, (Section 5.1) that aims at limiting these extra mispredictions through predicting branches with inflight non-retired occurrences: if the prediction for branch B is provided by the same predictor table and the same table entry that already provided the prediction for an already executed but not yet retired branch B' the we use the execution outcome of branch B' as a prediction for branch B rather than the output of the predictor. The IUM is shown to allow to recover almost all the mispredictions due to delayed updates. While TAGE is generally very efficient at predicting periodic regular behavior even when the period spans over hundreds or thousands of branches, it fails to predict loop exits if the control flow path inside the loop body is irregular. A loop predictor [6] can more efficiently track loops with constant numbers of iterations and therefore predict the exit of these loops with a very high accuracy (Section 5.2). On some applications, there exist branches (generally only a few) that are not strongly correlated with the branch history or path, but exhibit a statistical bias towards taken or not-taken. On these branches, the TAGE predictor generally performs worse than simpler solutions that capture statistical bias by using wide counters. The Statistical Corrector predictor (Section 5.3) is introduced to capture these branches and to predict them with an accuracy close to their statistical bias at a reasonable cost. The ISL-TAGE predictor combines TAGE with the IUM, the loop predictor and the Statistical Corrector Predictor.

In some applications, there remains some branches that are intrinsically better predicted through the use of local history than through the use of a global history. In the ISL-TAGE predictor proposal , the Statistical Corrector Predictor is indexed using the same global history as TAGE. We show that replacing the global history by the local history offers the opportunity to smoothly combine the TAGE predictor with local prediction. Associated with TAGE, a Statistical Corrector predictor based on local history (LSC for short) essentially captures these branches. It also captures most of the branches that the loop predictor is correctly predicting and most of the branches that the initial Statistical Corrector predictor based on global history is correcting. At similar storage budget, the TAGE-LSC (+ IUM) predictor outperforms the ISL-TAGE predictor [24] [1] using a smaller number of components and a smaller number of tagged tables, and without using any unrealistic tricks.

The remainder of the paper is organized as follows. Section 2 presents the experimental framework used in this paper. Section 3 recalls the main characteristics of the TAGE predictor used as the main predictor in this paper and presents its main qualities and its base performance. Section 4 analyzes the number of predictor table accesses on a real hardware processor and shows that the TAGE predictor exhibits marginal accuracy loss when the prediction is not reread at retire time, therefore potentially allowing simpler hardware implementation. Moreover we show that the TAGE predictor can be designed with single-port bank-interleaved memory components instead of 3-ported memory components. Section 5 presents the ISL-TAGE predictors. Section 5.1 introduces the Immediate Update Mimicker, IUM that recovers most of the mispredictions due to delayed update of the TAGE branch predictor tables. Section 5.2 describes the loop predictor and its benefit as a side predictor. Section 5.3 presents the Statistical Corrector predictor, that tracks branches that are not strongly correlated with the branch history path but are statistically biased. In Section 6, we show how the Statistical Corrector predictor can be leveraged to use local history in a side predictor for TAGE. We show that TAGE-LSC can achieve state-of-the-art prediction accuracy levels for the prediction storage budgets (128K-512Kbits) that are currently considered for implementation. In Section 7 presents a cost-effective implementation of the TAGE-LSC predictor using single-port memory components and avoiding predictor read at retire time can be avoided for correct prediction. Section 8 concludes this study.

2. Experimental Framework

In order to allow reproducibility of the experiments reported in this paper, the evaluation framework used is the one provided for

[1] winner of the 3rd Championship Branch Prediction

the 3rd Championship Branch Prediction, http://www.jilp.org/jwac-2/framework.html. The simulation is trace driven, but includes features to model a simple out-of-order execution core with a realistic memory hierarchy. It allows to delay branch prediction table updates till the retire stage in the pipeline. The benchmark set features 40 traces, approximately 50 million micro-ops long classified into 5 categories: CLIENT, INT (Integer), MM (Multimedia), SERVER and WS (Workstation). It is noticeable that some of these traces exhibit very large footprints (several tens of thousands of static branches), they also include both user and system activity.

2.1 Performance Metric

In order to remain consistent with the 3rd CBP, we will use the metric Misprediction Penalty per Kilo Instructions (MPPKI) proposed for the 3rd CBP. In practice, for the predictors considered in this paper, this metric is globally proportional to the misprediction number despite that the average misprediction penalty varies among the benchmarks.

2.2 Benchmark Set Characteristics

Based on a first analysis of the benchmark set after running a base 512Kbits L-TAGE predictor, the benchmark set can be divided in two major sets with very different misprediction rates.

A set of high misprediction rate benchmarks includes CLIENT0 INT01, INT02, MM05, MM07, WS03 and WS04. These 7 benchmarks represent approximately 3/4 th of the mispredictions of the 40 benchmarks. All these benchmarks exhibit more than 500,000 mispredictions (i.e more than 10 mispredictions per kilomicro-operations) for the reference predictor (see next section) considered in this study. Among this class, CLIENT02 is the only benchmark amenable to a low misprediction rate by simply increasing the size of the predictor to unrealistic storage budget (256 Mbits).

The set of the 33 medium and low misprediction rates benchmarks represent only 1/4th of the mispredictions and none of them exhibit more than 170,000 mispredictions on our reference predictor.

3. Background on the TAGE Predictor

The TAGE predictor was introduced in [26] and is the core predictor of the L-TAGE predictor that won the second Championship Branch Prediction in 2006 [23]. Figure 1 illustrates a TAGE predictor. The TAGE predictor features a base predictor T0 in charge of providing a basic prediction and a set of (partially) tagged predictor components Ti. The base predictor can be a simple PC-indexed 2-bit counter bimodal table. The tagged predictor components Ti, $1 \leq i \leq M$ are indexed using different global history lengths that form a geometric series, i.e, $L(i) = (int)(\alpha^{i-1} * L(1) + 0.5)$ as introduced for the OGEHL predictor [21]. On a TAGE predictor, most of the storage is used in tables indexed with (relatively) short histories, but the predictor is able to capture correlation with very old branches (up to 2000 branches in our experiments in this paper).

An entry of a tagged component of the TAGE predictor consists in a 3-bit prediction counter *ctr* whose sign provides the prediction, a (partial) tag and a useful bit *u* to guide replacement policy (Figure 2).

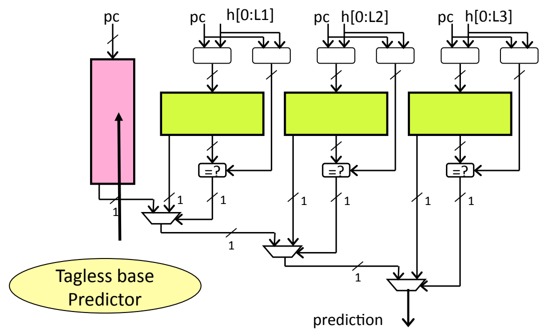

Figure 1. The TAGE predictor synopsis: a base predictor is backed with several tagged predictor components indexed with increasing history lengths

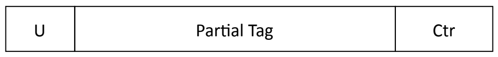

Figure 2. An entry in a TAGE tagged table

A few definitions and notations.
The *provider component* is the matching component with the longest history. The alternate prediction *altpred* is the prediction that would have occurred if there had been a miss on the provider component. If there is no hit on a tagged component then *altpred* is the default prediction.

3.1 Prediction Computation

At prediction time, the base predictor and the tagged components are accessed simultaneously. The base predictor provides a default prediction. The tagged components provide a prediction only on a tag match.

In the general case, the overall prediction is provided by the hitting tagged predictor component that uses the longest history, or in case of no matching tagged predictor component, the default prediction is used. It was remarked that when the provider component is a tagged component and the prediction is weak, the confidence in the prediction is quite low (often less than 60%). In this situation, the alternate prediction is often more accurate than the provider component prediction. This property was found to be essentially temporal on the whole application. Dynamically monitoring it through a single 4-bit counter USE_ALT_ON_NA was found to allow to (slightly) improve prediction accuracy [26]. The prediction computation algorithm is as follows:

1. Find the matching component with the longest history
2. if (the prediction counter is not weak or USE_ALT_ON_NA is negative) then the prediction counter sign provides the prediction else the prediction is the alternate prediction

3.2 Predictor Update

In all cases, the prediction counter of the provider component is updated. The useful bit *u* is set if the prediction was correct *and* the alternate prediction *altpred* was incorrect. New tagged entries are allocated only on mispredictions.

3.2.1 Allocating tagged entries on mispredictions

In previous studies on the TAGE predictor, it was argued to allocate at most one entry to limit the footprint of the application on the predictor. In practice, the footprint limitation holds only on demanding applications and for small predictors (e.g. 64 Kbits) or predictors with a small number of tables. For large TAGE predictors (in the 256Kbits-512Kbits range), we discovered that allocating several (in practice up to 3 or 4) entries in different tables enables a very short warming or transition phase in the predictor. It also allows us to use a very simple and effective way of managing the useful bits on the predictor (see below in Section 3.2.2).

If the provider component Ti is not the component using the longest history (i.e., $i \leq M$), then up to four entries on a predictor component Tk with $i < k \leq M$ are allocated on non-consecutive tables. The entries are chosen among the useless entries, i.e., with a null *u* bit. An allocated entry is initialized with the prediction counter set to weak correct. Bit *u* is initialized to 0 (i.e., *not useful*).

3.2.2 Managing the *u* bit

The useful bit *u* is used for locking entries in the predictor. The *u* bit of a provider component is set whenever the prediction was correct *and* the alternate prediction *altpred* was incorrect. In order to avoid the useful bits to stay forever set, the following resetting policy is implemented. The number of successes and failures on entries allocations is monitored; this monitoring is performed through a single 8-bit counter (u=1, increment, u=0, decrement). This counter saturates when more failures than successes are encountered on allocations. At that time we reset all the *u* bits of the predictor. Typically, such a global reset occurs when one out of two entries on the used portion of the predictor has been set to useful.

This simple policy was found to be more efficient than the previously proposed management using 2-bit useful counters for the TAGE predictor [26][23]; it also uses a single *u* bit per tagged TAGE entry.

3.3 Implementation Tradeoffs

Various parameters can be tuned on the TAGE predictor in order to get prediction accuracy.

Depending on the storage budget one will use small or large number of tables. It was shown in [23] that for small budgets (e.g. 32K-64Kbits) using 5 to 8 tables is good tradeoff and that for large budgets (e.g. 256Kbits-512Kbits) using 12-15 tables is a better tradeoff.

Most of the storage budget in the TAGE predictor is dedicated to the partial tags. Using a large tag width leads to waste part of the storage while using a small tag width leads to false tag match detections. We found that , for a 13 tables TAGE predictor using 12-bit tag is good tradeoff. However Experiments showed that one can use wider tag for long histories for a better tradeoff. Previous experiments [26] have shown that the TAGE predictor performs efficiently on a wide spectrum of history lengths with maximum history lengths ranging from a 100 to more than 1,000 history bits.

3.4 The Reference TAGE Predictor

This study targets the 64Kbytes storage budget that was allowed for the 3rd Championship Branch Prediction. Therefore the reference TAGE predictor that will be used in this study has been dimensioned to fit in that storage budget. It features a 13-component, and uses the (6,2000) geometric history length with the following table sizes for a total of 65,408 bytes of storage:

- Bimodal table: 32K prediction bits + 8K hysteresis bits
- Tag width for table Ti: max (6+i, 15) bits
- Table T1, 2Kentries, Tables 2-7, 4Kentries, Tables 8-9, 2Kentries and Tables 10-12, 1K entries

4. The Delayed Update Issue on Conditional Branch Predictor

On a real hardware processor, the predictor tables are updated at retire time to avoid pollution of the predictor by wrong path branches. A single predictor table entry may provide several mispredictions in a row due to this late update. While this phenomenon has been recognized by the microarchitecture research community, it has been considered to be relatively marginal on global history predictors featuring long history [10, 21]. However these studies were assuming that the predictor tables were reread and the prediction recomputed at retire time: each branch on the correct path potentially leads to 3 accesses of the predictor tables, read at prediction time, read at retire time and write at retire time.

At a first glance, the second read at retire time appears useless since one could use the values read at prediction time. However the simple example of a loop execution and a bimodal 2-bit counter predictor illustrated on Figure 3 shows that this second read might be really needed. On the example the first 6 iterations are mispredicted: till retiring the first iteration, i.e. for iterations 2 and 3, the value read on the predictor is C=0, i.e. strongly not-taken, then till retiring iteration 4, i.e. iterations 4, 5 and 6, the value read on the predictor is C=1, i.e weakly taken; the bimodal predictor predicts correctly at iteration 7 instead of of iteration 3 when immediately updated or iteration 5 if the predictor was reread at update time.

Three predictor accesses per branch would normally necessitate the use of 3-port predictor tables in order to issue and to commit one branch per cycle. Unfortunately 3-port memory component occupy much more area and dissipates much more energy than single ported memory component. Simple experiments run on CACTI 6.5 [19] showed that, for the range of memory array sizes used in branch predictors (1Kbytes to 64Kbytes) and for equal capacity the area of a 3-port memory array is 3-4 times larger than a single-ported memory array, while the energy dissipated per access is about 25-30 % higher.

To the best of our knowledge, no constructor has ever disclosed how these potential 3 accesses per branch on the predictor are handled in the hardware implementation of the predictor when one want to sustain one branch prediction per cycle. Since 3-port predictor tables are not hardware cost-effective, alternative techniques are needed.

In the remainder of the section, we present realistic techniques to reduce the number of accesses on the branch predictor, particularly for the TAGE predictor. Such techniques can be leveraged in real designs to build predictors using dual-ported or single-ported

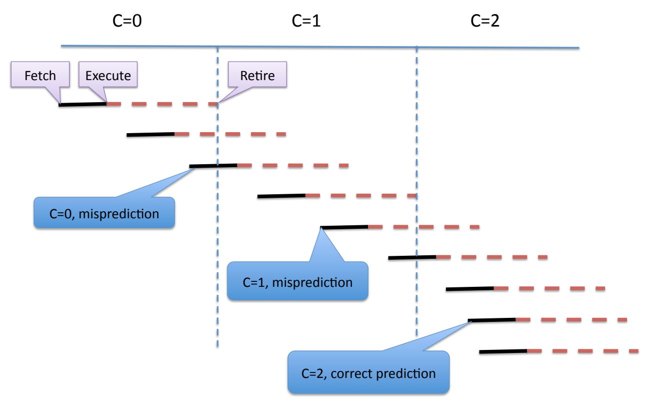

Figure 3. Example of a loop execution assuming a bimodal table with null prediction counter at the first iteration.

memory components. More precisely, in Section 4.3, we propose an implementation of the TAGE predictor using only single-ported memory arrays.

4.1 Reducing the Number of Accesses on the Branch Predictor

4.1.1 Eliminating silent updates

The first obvious technique to reduce the number of accesses on the branch predictor is to avoid the silent updates of the predictor. On update, in many cases the table entries are overwritten with the exact same value they were previously holding. Eliminating this silent update is a simple technique that holds for all branch predictors and that should be very effective. This technique was already mentioned by Baniasadi and Moshovos in the context of energy saving [2].

We confirmed this assumption by simulations of 3 very different predictors: our reference TAGE predictor, a simple 512 Kbits *gshare* [16] predictor as a representative of first generation predictor and a 520 Kbits GEHL predictor [21] as a representative of neural inspired predictors [2]. A GEHL predictor featuring 13 tables, 5 bit entries and 8K entries per table using (6,2000) history length , i.e. a total of 520 Kbits was considered [3].

In practice, on the benchmark set, TAGE encounters an average of 2.17 effective writes per misprediction or 9.06 writes per 100 retired conditional branches. GEHL, by construction of its update policy, encounters slightly less than 2 effective writes per misprediction (1.94) or 9.10 writes per 100 retired branches. *Gshare* encounters in average 1.54 writes per misprediction or 9.61 writes per 100 retired branches.

[2]by neural inspired, we mean that prediction is computed through a tree of adders and that the predictor update is threshold based as described in the initial perceptron predictor study [13]

[3]The GEHL configuration was not optimized. The purpose of this experiment is not here to compare accuracies of GEHL and TAGE, but to illustrate the issues of delayed update and multiple accesses to the predictor tables on different predictors

4.1.2 Eliminating the read at retire time ?

One would like to avoid the read of the predictor tables at retire time if possible. However there might be performance loss. We have conducted experiments to evaluate three possible scenarii on the 3 branch predictors used above and compared their performance with the oracle update at fetch time scenario **[I]** as a reference. The three possible scenarii are:

1. **[A]**: reread the prediction tables at retire time before (potential) update.
2. **[B]**:only read prediction at fetch time, propagate all the informations with the branches in the pipeline, and write the update at retire time.
3. **[C]**: reread the predictions at retire time for mispredicted branches only.

Scenario **[A]** potentially leads to 3 accesses per branch prediction and systematically generates 2 reads for each retired branch. Scenario **[B]** generates at most one read and one write. Scenario **[C]** generates 2 read accesses only on mispredicted branches. Scenarii **[B]** and **[C]** necessitate that the values read on the predictor are propagated with the branch instruction in the pipeline and used at retire time.

In practice for the first generation predictors such as *gshare* using a single table of prediction counters, the accuracy loss encountered in Scenario **[B]** is dramatic. On the 512 Kbits *gshare*, we respectively obtained **[I]** 944 MPPKI, **[A]** 970 MPPKI , **[B]** 1292 MPPKI , **[C]** 1011 MPPKI. Scenario **[B]** where the predictor table is only read at fetch time can not be considered as a solution for a real hardware implementation. Even Scenario **[C]** , rereading the predictor at retire time for mispredicted branches, leads to a quite significant accuracy loss.

On predictors derived from neural-inspired predictors (perceptron, piecewise linear, OGEHL), the same phenomenon occurs as illustrated on the 520 Kbits GEHL predictor. The different scenarii lead to the following results : **[I]** 664 MPPKI , **[A]** 685 MPPKI , **[B]** 801 MPPKI, **[C]** 744 MPPKI. While the 3.5 % extra penalty due to delayed update is acceptable in Scenario **[A]** , the losses due to the use of old counter values on correct predictions (Scenarii **[B]** and **[C]**) are very large and cannot be considered for effective implementation. On the GEHL predictor, 13 counters are involved in a prediction. In particular when the same branch has several inflight occurrences, the prediction counters associated with the short histories are not correctly incremented or decremented.

For the TAGE predictor, the accuracy loss phenomenon is much less pronounced: **[I]** 609 MPPKI, **[A]** 617MPPKI, **[B]** 640 MPPKI, **[C]** 625 MPPKI. On TAGE, on correct predictions, only one or two TAGE entries are involved in a prediction and its update. On a correct prediction, in Scenarii **[B]** and **[C]** , the update will use old prediction counter values only if the provider component and entry was updated by an already inflight branch at prediction time. Moreover, at the difference of the neural inspired predictors, the amplitude of the prediction counters is limited.

4.2 Summary on Number of Predictor Accesses per Branch

Compared with predictors using wide counters or predictors using a single predictor table, TAGE suffers only a marginal accuracy impact when the predictor table is read only once at prediction time particularly if it is read at retire time on mispredictions. The performances losses on Scenario **[C]** on GEHL predictors and gshare will push designers to reject this scenario and therefore to perform systematically two reads on the predictor tables per branch.

On the other hand, the validity of Scenario **[C]** for TAGE opens the possibility to implement the TAGE predictor with bank-interleaved tables built with single-ported memory banks (e.g. 4-way interleaved) since each (retired) branch generates in average only 1.13 accesses (1 read at prediction time, 0.04 read in average at retire time, and 0.09 write in average at retire time) to the predictor tables. Such a solution can leverage the proposition for the EV8 branch predictor [22] to guarantee that two or even three consecutive branch predictions will access different banks in each table as shown below.

4.3 Towards 4-way Interleaved single-ported Predictor Tables for TAGE

In this section, we assume that each of the tables in the TAGE predictor is 4-way interleaved. We assume that a single branch (either direct or indirect) is predicted per cycle. In order to guarantee a smooth repartition of the accesses among the banks, one can guarantee that the predicted branch will not access any of the banks that were accessed by the two previous predictions as follows:

Let X,Y and Z be the PCs of the three successive branches, we compute b(Z), the number of the bank accessed for the prediction of Z as follows:

```
if (Z is unconditional) b(Z) = -1 ; /*i.e. no access*/
else
{b(Z) = Z & 3; while ((b(Z) == b(X)) || (b(Z) == b(Y))
b(Z) = (b(Z) +1) & 3;}
```

We assume that prediction has priority over update and that write at retire time has priority over read at retire time. Our algorithm guarantees that for every bank of any of the predictor tables, for every 3 cycles interval, 2 cycles are free for performing the updates.

Since for scenario **[C]**, reads at retire time and effective write updates are rare, no huge buffering is needed. In practice, reading the predictor, computing the update and updating the predictor necessitates a few cycles: the read at retire time can be delayed by one cycle (conflict with a prediction) and the update by up to two cycles (conflict with prediction or/and an older update).

Simulations of bank-interleaving for scenario **[C]** leads to 627 MMPKI, i.e., very close to the 625 MPPKI achieved without bank-interleaving, but using 3-port memory arrays. Evaluations using CACTI 6.5 [19] report a 3.3x decrease of the silicon area occupied by the memory arrays and a 2x decrease of the energy dissipated in the memory arrays per predictor access when assuming bank-interleaving instead of 3-port memory array.

Note that this bank-interleaving technique can be applied for most global history predictors also resulting in marginal accuracy losses. On local history predictors, the bank-interleaving technique can also be applied on the predictor tables but results in more substantial prediction accuracy loss since several entries correspond to the same (history,branch) pair, thus inducing more aliasing conflicts, and necessitating a longer training.

In the remainder of the paper apart Section 7, we will consider that predictors are implemented using 3-port memory arrays. Moreover apart Section 5.1 and Section 7, we will consider that the predictor is systematically reread at retire time.

5. The ISL-TAGE predictor

In this section, we detail the side predictors that were proposed for TAGE at the 3rd Championship on Branch Prediction [24].

5.1 The Immediate Update Mimicker

In the previous section, we have pointed out that the impact of delayed update on the TAGE predictor is much lower than on other predictors including the neural-inspired predictors, particularly when one try to only read at prediction time and write at retire time. In order to further limit the impact of the use of the delayed update, an add-on to TAGE can be implemented, the Immediate Update Mimicker, IUM.

On a misprediction, the global history can be repaired immediately and when a block is fetched on the correct path, the speculative branch history is correct. In practice, repairing the global history is straightforward if one uses a circular buffer to implement the global history. We leverage the same idea with IUM predictor (Figure 4).

When fetching a conditional branch, IUM records the prediction, the identity of the entry E in the TAGE predictor (number of the table and its index) that provides the prediction. At branch resolution on a misprediction, the IUM is repaired through reinitializing its head pointer to the associated IUM entry and updating this entry with the correct direction.

When fetching on the correct path, the associated IUM entry associated with an inflight branch B features the matching predictor entry E that provided the TAGE prediction and the **effective** outcome of branch B (corrected in case of a misprediction on B). In case of a new hit on entry E in the predictor before the retirement of branch B, the (TAGE predictor + IUM) can respond with the direction provided by the IUM rather than with the TAGE prediction (on which entry E has not been updated).

IUM can be implemented in hardware through a fully-associative table with one entry per inflight branch. It allows to recover about 3/4th of the mispredictions due to late update of the TAGE predictor tables if all predictions are recomputed are retire time (Scenario **[A]**, 611 MPPKI vs Scenario **[I]** 609 MPPKI). For Scenario **[B]**, prediction tables are read only at fetch time, half of the mispredictions due to delayed updates are recovered (624 MPPKI) while for Scenario **[C]** where prediction tables are reread only on mispredictions, the accuracy loss is limited (614 MPPKI).

Without the IUM, the gap between immediate update and realistic update is very dependent on the benchmarks. For some benchmarks, particularly on the set of high misprediction rate benchmarks and for scenarii **[A]** and **[C]** the difference between immediate update and realistic update is marginal ($< 1\%$ of the mispredictions), while on other benchmarks (e.g. CLIENT04, CLIENT06) it represents more than 10 % of the mispredictions. For all the benchmarks, the IUM reduces this gap to less 1% for Scenario **[A]** or 2% for Scenario **[C]** . For Scenario **[B]**, CLIENT02, a high misprediction rate benchmark, is a specific outlier with an accuracy loss of 9 % without the IUM. With the IUM, it reaches an acceptable 5 % accuracy loss.

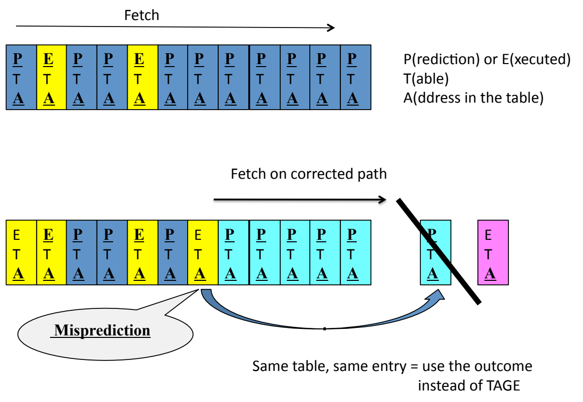
Fetch
P(rediction) or E(xecuted)
T(able)
A(ddress in the table)
Fetch on corrected path
Misprediction
Same table, same entry = use the outcome instead of TAGE

Figure 4. The Immediate Update Mimicker: when the prediction is provided by the same entry and the same component as an already executed inflight branch, use the outcome of the branch instead of the TAGE prediction.

Note that for implementing Scenario **[B]** or **[C]** in a real hardware predictor, one will have to propagate information read on the predictor at prediction time up to retire time. The IUM entry can be augmented with a few fields to propagate these information.

For the sake of simplicity and since the accuracy difference between Scenario **[A]** and Scenario **[C]** is very limited, we will use Scenario **[A]** in the remainder of the paper.

5.2 The Loop Predictor

Loops with constant number of iterations are branches that are highly predictable when one only considers the local history. The TAGE predictor is generally able to predict these loops with very high accuracy when the control flow inside the loop is regular. However when the control flow in the loop body is erratic, the TAGE predictor may fail to correctly predict the exit of the loop [6].

A loop predictor can simply identify regular loops with constant number of iterations. The loop predictor will provide the global prediction when it identifies the branch as a loop with a constant iteration number and when this identificatiion has reached a high confidence, i.e. when the loop has been executed several times with the same number of iterations. In practice, reaching a high confidence level after 7 executions of the overall loop appears as a good tradeoff.

A loop predictor with a limited number of entries and high associativity is sufficient. In the experiments we report below, a 4-way skewed associative 64-entry loop predictor is assumed. Each entry in the loop predictor table consists of a past iteration count on 10 bits, a retire iteration count on 10 bits each , a partial tag on 10 bits, a confidence counter on 3 bits, an age counter on 3 bits and 1 direction bit i.e. 37 bits per entry. Replacement policy is based on the age. An entry can be replaced only if its age counter is null. On allocation, age is first set to 7. Age is decremented whenever the entry was a possible replacement target and incremented when the entry is used and has provided a valid prediction **and** the prediction would have been incorrect otherwise. Age is reset to zero whenever the branch is determined as not being a regular loop.

The overall hardware complexity of the loop predictor is not in the loop predictor table itself but in the speculative management of the iteration numbers.

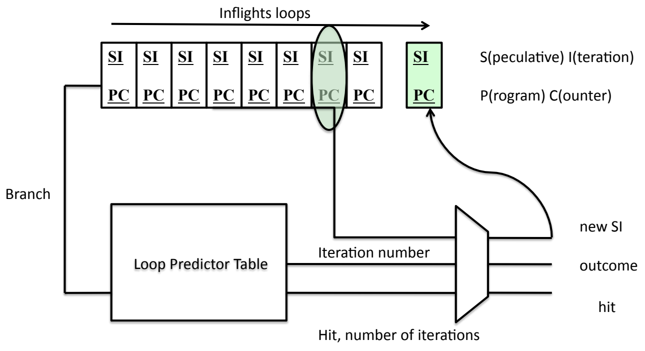

Figure 5. The loop predictor and the Speculative Loop Iteration Management: use of the non-speculative iteration number or of the iteration number of the most recent inflight iteration.

Figure 5 illustrates here a possible implementation of such a speculative management, a Speculative Loop Iteration Manager, SLIM. The SLIM records a new entry for each branch recognized as a loop. The entry features the PC of the branch and the (speculative) iteration number that has been reached. At prediction time, the loop predictor is checked. On a hit with high confidence, the (non-speculative) number of the current iteration and the number of iterations in the loop are read on the predictor. The SLIM is read in parallel. If the branch also hits in the SLIM then the most recent hitting entry provides the speculative iteration number. The speculative iteration number is incremented and checked against the predicted number of iterations of the loop, branch outcome is predicted accordingly. On a misprediction, the SLIM entries after the misprediction are cleared. On a loop retirement, the associated SLIM entry is cleared.

Associating the loop predictor on top of the base TAGE+IUM predictor allows to reach 593 MPPKI, i.e. approximately a 3 % reduction of the performance loss due to imperfect prediction by the TAGE+IUM predictor.

5.3 The Statistical Corrector Predictor

The TAGE predictor is very efficient at predicting very correlated branches even if the correlation is with very remote branches, e.g. on a 1000 bits branch history. However, TAGE fails at predicting statistically biased branches e.g. branches that are not correlated with the control flow path, but have only some statistical bias towards a direction. On some of these branches, the TAGE predictor performs even worse than a simple PC-indexed table of wide counters (e.g. 5 bit counters).

In order to better predict this class of statistically biased branches, the Statistical Corrector predictor is introduced [24]. The correction aims at detecting the unlikely predictions and to revert them: the prediction and the (address, history) pair is presented to Statistical Corrector predictor which decides whether or not inverting the prediction (similar to the Agree predictor [27]). Since in most cases the prediction provided by the TAGE predictor is correct, the Statistical Corrector predictor agrees most of the time with the TAGE predictor. Therefore a relatively small Statistical Corrector predictor performs close to an unlimited size Statistical Corrector predictor.

A quite efficient implementation of the Statistical Corrector predictor was derived from the GEHL predictor [21]. It features 4 logical tables indexed with the 4 shortest history lengths (0, 6, 10, 17) as the main TAGE predictor **and** the prediction (Taken/Not taken) flowing out from the TAGE predictor. The tables are 1K 6-bit entries, i.e., a total of 24 Kbits.

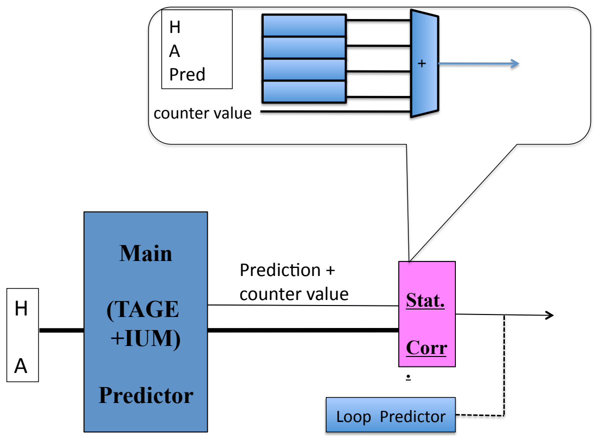

Figure 6. The Statistical Corrector predictor

The prediction is computed as the sign of the sum of the (centered) predictions read on the Statistical Corrector table plus eight times the (centered) output of the hitting bank in TAGE. By centered predictions, we mean 2*ctr+1. Incorporating the prediction counter value from TAGE allows to take into account the confidence in the TAGE prediction [25]. The TAGE prediction is reverted if the Statistical Corrector predictor disagrees **and** the absolute value of the sum is above a dynamic threshold. The dynamic threshold is adjusted at run-time in order to ensure that the use of the Statistical Corrector predictor is beneficial. The technique used to adapt the threshold is similar to the technique proposed for dynamically adapting the update threshold of the GEHL predictor [21].

When adding the Statistical Corrector predictor on top of the TAGE + IUM + loop predictor, a performance of 580 MPPKI is reached, i.e. approximately a 2 % reduction of the performance loss due to imperfect TAGE+IUM+loop predictor. Unrealistic tricks were used in [24] to bring the performance to 568 MPPKI.

5.4 The ISL-TAGE Predictor Performance

Using side predictors of relatively small size, the ISL-TAGE predictor reduces the misprediction rate of the 512Kbits TAGE predictor by 6 % which is approximately the misprediction reduction that would be obtained by scaling the TAGE predictor to 2Mbits.

6. TAGE-LSC: Adapting the Statistical Corrector Predictor to Local History

In many applications, there exist branches whose behaviors are only correlated with their own local history (loops, but also periodic behavior). Most of these behaviors are captured by the global history predictors, but some are not. Hybrid predictors combining local history and global history were introduced [16] for capturing the best of global correlation and local correlation. However metapredictors were shown to be poorly efficient and new solutions combining local history and global history were proposed for neural based predictors [12, 8]. Up to now, there has not been any elegant proposal to combine the global history TAGE predictor with some efficient local history predictor. The approach used for the Statistical Corrector Predictor can be adapted to offer such a solution as illustrated in Figure 7.

6.1 Architecture of the TAGE-LSC predictor

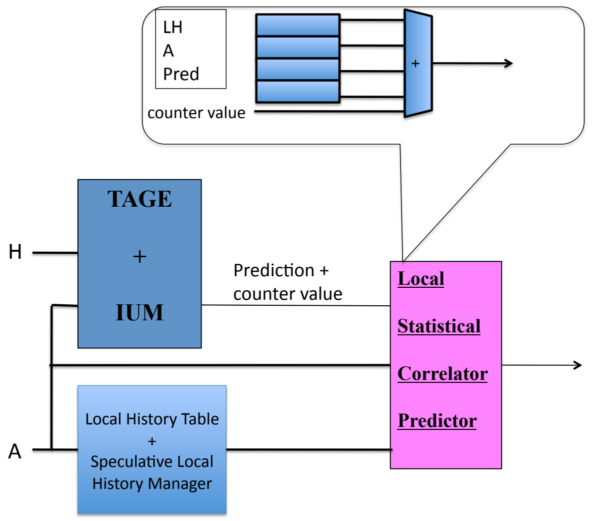

Figure 7. The TAGE-LSC predictor combines a TAGE predictor and a Local history Statistical Corrector Predictor

On the LSC predictor (Figure 7), a local history table and a Speculative Local History Manager (Figure 8) are accessed in parallel with the TAGE predictor. When a branch with same index in the local history table is already inflight the pipeline, the Speculative Local History Manager provides the speculative local history. The speculative local history is associated with the TAGE prediction to index a LGEHL (Local history GEHL) predictor. The prediction is computed as the sign of the sum of the (centered) predictions read on the Statistical Corrector table plus eight times the (centered) output of the hitting bank in TAGE. The TAGE predic-

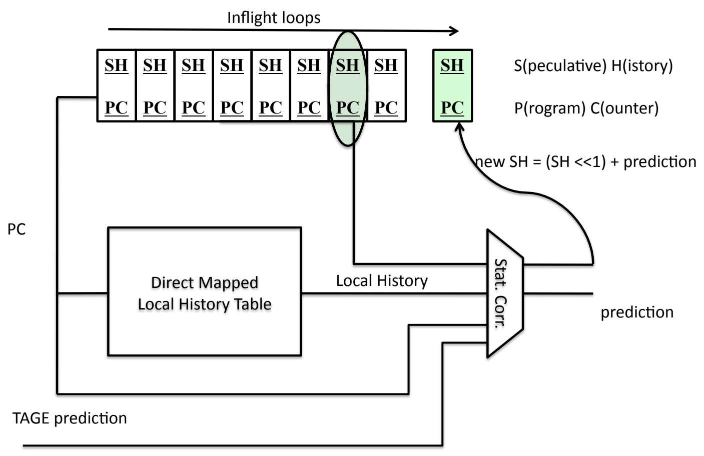

Figure 8. Speculative Local History Management

tion is reverted if the Statistical Corrector predictor disagrees **and** the absolute value of the sum is above a dynamic threshold.

The structure of the Speculative Local History Manager is very similar to the Immediate Update Mimicker (one entry per inflight branch, associative search on inflight branches, . . .). Therefore these two structures could be combined in a real hardware processor

We found that using 5 tables featuring 1K 6-bit entries and history lengths (0,4, 10, 17, 31) on the LGEHL predictor and a small 32-entry direct-mapped local history table is sufficient to capture most of the local correlation that a 512 Kbits TAGE predictor does not capture on our benchmark set. In most cases, the LSC prediction just agrees with the TAGE prediction, but when it disagrees and above the threshold, it provides the correct prediction in more than 70 % of the cases.

The very small size of the required local history table can be explained by the observation that for most benchmarks a very small number of static branches (generally less than 10) represent the majority of the dynamic mispredictions on the TAGE predictor.

On top of the (TAGE + IUM + loop predictor + Statistical Corrector predictor), the LSC predictor allows to reach 555 MPPKI, thus nearly reducing misprediction rate by more than 4%. In practice, using the LSC predictor alone on top the TAGE+IUM predictor allows to reach 559 MPPKI, i.e. it captures most of the mispredictions that are captured the loop predictor and the Statistical Corrector predictor. A 30 Kbits LSC predictor reduces the performance loss due to imperfect prediction of the TAGE+IUM predictor by more than 8 %.

In order to get fair comparison with the ISL-TAGE [24], we adjusted the size of the TAGE-LSC predictor to 512 Kbits by simply reducing the size of Table T7 to 2K entries (thus saving 34K storage bits) in the reference predictor. We obtain 562 MMPKI against 581 MPPKI for a 512 Kbits ISL-TAGE predictor with similar structure (same TAGE predictor, 5 tables GEHL-like predictor for Statistical Corrector Predictor **and** loop predictor) and 568 MPPKI for the ISL-TAGE predictor presented at the 3rd CBP (16-component TAGE+ IUM+ Statistical Corrector predictor + loop predictor + costly tricks such as sharing/interleaving TAGE tagged tables, sharing/interleaving Statistical Corrector predictor tables and extensive search for the best set of history lengths).

6.2 Performance Evaluation of the TAGE-LSC Predictor

Many parameters can be varied in the TAGE-LSC (storage size, number of tables and history lengths).

As its parents TAGE and GEHL predictors, the TAGE-LSC predictor is very robust to the choice of the history lengths, for instance on a 512Kbits TAGE-LSC predictor, using a (3,300) history series lead to 575 MPPKI, the (4,1000) history series lead to 563 MPPKI, and the (8,5000) history series leads to 563 MPPKI.

Decreasing the number of the TAGE predictor tables has not a large impact. A 9-component TAGE-LSC predictor using (6,1000) history series achieves 566 MPPKI and a 6-component TAGE-LSC predictor using (6,500) history series achieves 583 MPPKI.

On Figure 9, we illustrate the respective performance for TAGE predictors and TAGE+LSC predictors for size ranging from 128 Kbits to 32 Mbits. The simulations were run just by scaling the sizes of all the components by a power of two, no attempt to optimize other parameters was done. For both predictors, the performance reaches a plateau around 16-32Mbits. In fact for 39 out of the 40 benchmarks, it reaches the plateau well before 16-32 Mbits, but for CLIENT02 the misprediction rate suddenly falls at 2Mbits- 8Mbits budget: most of the mispredictions are due to only 2 branches that have repetitive behaviors but with thousands of different patterns. The TAGE-LSC predictor exhibits more potential than the TAGE predictor. In the range of size of interest for hardware implementation, 128Kbits - 512Kbits, the TAGE-LSC predictor performs consistently as a 4-8 larger TAGE predictor.

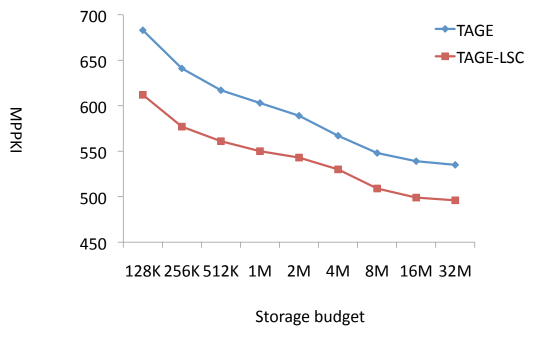

Figure 9. TAGE vs.TAGE-LSC

6.3 Comparisons with Alternative Branch Predictors

At the 3rd ChampionShip Branch Prediction, the FTL++ predictor [7] and the OH-SNAP [14] were respectively ranked 2nd and 3rd with respectively 581 MPPKI and 598 MPPKI. The configurations presented at the Championship have some features that are not realistically implementable. However both these predictors capture some correlations that are not captured by TAGE-LSC.

Both FTL++ and OH-SNAP are based on neural based techniques, respectively GEHL combined with LGEHL for FTL++ and piecewise linear [11] combined with dynamic weight adaptation [1] for OH-SNAP.

Both these predictors are significantly outperformed on the 33 most predictable benchmarks by both ISL-TAGE and TAGE-LSC: respectively 196 MPPKI for ISL-TAGE, 198 MPPKI for TAGE-LSC, 232 MPPKI for FTL++ and 254 MPPKI for OH-SNAP.

On the other hand, both FTL++ and OH-SNAP slightly outperform ISL-TAGE and TAGE-LSC on the 7 most unpredictable benchmarks as illustrated by Figure 10: respectively 2311 MPPKI for ISL-TAGE, 2287 MPPKI for TAGE-LSC, 2222 MPPKI for FTL++ and 2227 MPPKI for OH-SNAP. Understanding the intrinsic auto- and inter-correlation properties that are better captured by FTL++ and/or OH-SNAP than TAGE-LSC might lead to further progress in branch prediction research.

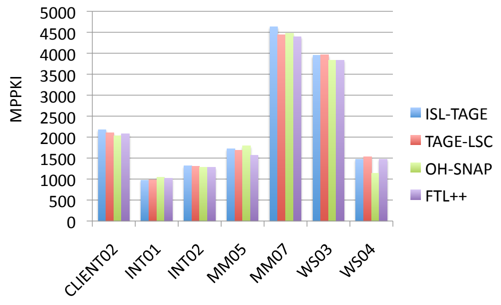

Figure 10. Prediction accuracy on the 7 less predictable benchmarks

7. Towards Cost-effective TAGE-LSC Predictor

In Section 4, we have shown that, at a marginal accuracy loss the TAGE predictor can be implemented using 4-way bank-interleaved single port memory components and that the predictor read at retire time can be avoided on correct predictions also at a marginal accuracy loss.

However the TAGE-LSC predictor features local history components which accuracy could be impaired when applying these optimizations. Therefore we run experiments with the TAGE-LSC predictor to assess the accuracy loss due applying the same complexity reduction to the local history components on the complete TAGE-LSC predictor.

7.1 Bank interleaving

Using the bank-interleaving technique described in Section 4.3 on local history components can have two conjugated impacts on the prediction accuracy. First more entries are needed for mapping a single branch (up to four times), second more entries have to be trained, thus leading to more start-up mispredictions.

Using bank-interleaving on the 512 Kbits TAGE-LSC considered in the previous section revealed to be quite efficient even for the local history components apart for a single application trace CLIENT02. Doubling the number of entries in the local components was found to restored the accuracy on CLIENT02. In our experiments, we were able to define a 512 Kbits TAGE-LSC predictor using 4-way interleaved single-ported tables and achieving 569 MPPKI, i.e. an accuracy loss of 3 MPPKI due to interleaving on the local components (training impact), 2 MPPKI due to interleaving on the TAGE components and 2 MPPKI due to size reduction of some tables of TAGE to fit the 512 Kbits storage budget.

Evaluations with CACTI6.5 [19] indicate that this interleaving allows to reduce the silicon area by approximately a factor 3.3 and to approximately halve the power consumption per predictor read access.

7.2 Eliminating the Read at Retire Time on Correct Predictions

In Section 4, we have shown that the second read at retire time can be avoided on the TAGE predictor when prediction is correct with limited impact on the predictor accuracy.

On top of the 4-way interleaved TAGE-LSC predictor considered above, applying this optimization on both the TAGE components and the LSC components does not degrade too much the accuracy, since 575 MPPKI is achieved. This accuracy loss is very limited (2 MPPKI) if the optimization is applied only applied only to the TAGE components. Eliminating the second read for correct predictions on the local history components has a slightly higher accuracy impact (4 MPPKI). Since silent updates represent the vast majority of updates and can be eliminated, the elimination of the read at retire time on correct prediction allows to nearly halves the energy consumption of the predictor on correct predictions.

On the other hand, completely eliminating the second read (Scenario [B] in Section 4) has as a much higher impact accuracy impact (599 MPPKI) and therefore is not recommended.

8. Conclusion

The TAGE predictor has often been considered as state-of-the-art in conditional branch prediction in terms of prediction accuracy [1]. Asserting confidence to predictions by TAGE has recently been shown to be simple and storage free [25]. In this paper, we have further made the case for considering the TAGE predictor in real hardware processors through two directions: 1) reduction of its implementation cost and its energy consumption 2) improvement of its prediction accuracy.

On a processor, a prediction on a correct branch generates 3 predictor table accesses, read at prediction time, second read at retire time and write update at retire time. This normally leads to the use of multiport memory components for the branch predictor. We have shown that, on a real hardware implementation, the

TAGE accuracy would not be significantly impaired by avoiding a systematic second read of the prediction tables at retire time for correct prediction. Combined with the elimination of silent updates, this leads to only 1.13 predictor tables accesses in average per retired branch. Furthermore, we have shown that it is possible to implement the TAGE predictor tables as bank-interleaved structures single-port banks, thus significantly reducing the silicon footprint of the predictor as well as its energy consumption.

We have also shown that to further improve TAGE predictor accuracy, TAGE can be augmented with (small) side predictors, each targeting a set of branches that are difficult to predict with TAGE. We have presented the Immediate Update Mimicker that tracks the inflight already executed but not retired branches and uses their result for correcting the predictions. The IUM allows to recover most of the mispredictions due to delayed updates. As already proposed for the L-TAGE [23], TAGE can be augmented with a loop predictor to correctly predict loops with constant iteration numbers. TAGE is very efficient at predicting the behavior of branches that are completely correlated with the path history even when this path is thousands of branches long. But TAGE fails at capturing branches that are not strongly biased but that are only statistically biased. To capture these statistical biases, we have introduced the Statistical Corrector Predictor. Relying on the use of wide counters, the Statistical Corrector predictor tracks these statistically correlated branches. These 3 side predictors were presented at the 3rd Championship Branch Prediction for the ISL-TAGE predictor [24]. We have further extended this approach showing that the Statistical Corrector Predictor can be used to leverage local history to further improve the potential of the TAGE predictor. Furthermore,the LSC, Statistical Corrector predictor based on local history, dwarfs the benefits of the loop predictor and the global history Statistical Corrector and therefore makes a good case for the use of a TAGE-LSC predictor which combines a main TAGE predictor with IUM and a small LSC, Local history Statistical Corrector predictor.

Finally we have shown that the TAGE-LSC predictor is amenable to a cost-effective implementation at a marginal accuracy loss through exploiting bank-interleaved structures and avoiding the second read at retire time on correct predictions.

9. References

[1] Renée St. Amant, Daniel A. Jiménez, and Doug Burger. Low-power, high-performance analog neural branch prediction. In *MICRO*, pages 447–458, 2008.

[2] Amirali Baniasadi and Andreas Moshovos. Sepas: a highly accurate energy-efficient branch predictor. In *ISLPED*, pages 38–43, 2004.

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

[4] 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 *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996.

[5] Hongliang Gao, Yi Ma, Martin Dimitrov, and Huiyang Zhou. Address-branch correlation: A novel locality for long-latency hard-to-predict branches. In *HPCA*, pages 74–85, 2008.

[6] Hongliang Gao and Huiyang Zhou. Adaptive information processing: An effective way to improve perceptron predictors. *Journal of Instruction Level Parallelism (http://www.jilp.org/vol7)*, April 2005.

[7] Y. Ishii, K. Kuroyanagi, T. Sawada, M. Inaba, and K. Hiraki. Revisiting local history for improving fused two-level branch predictor. In *Proceedings of the 3rd Championship on Branch Prediction, http://www.jilp.org/jwac-2/*, 2011.

[8] Yasuo Ishii. Fused two-level branch prediction with ahead calculation. *Journal of Instruction Level Parallelism (http://wwwjilp.org/vol9)*, May 2007.

[9] D. Jimenez. Fast path-based neural branch prediction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, dec 2003.

[10] D. Jiménez. Reconsidering complex branch predictors. In *Proceedings of the 9th International Symposium on High Perform ance Computer Architecture*, 2003.

[11] D. Jiménez. Piecewise linear branch prediction. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, june 2005.

[12] D. 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*, Monterey, California, December 2000.

[13] D. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the Seventh International Symposium on High Perform ance Computer Architecture*, 2001.

[14] Daniel A. Jiménez. Oh-snap: Optimized hybrid scaled neural analog predictor. In *Proceedings of the 3rd Championship on Branch Prediction, http://www.jilp.org/jwac-2/*, 2011.

[15] 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.

[16] S. McFarling. Combining branch predictors. TN 36, DEC WRL, June 1993.

[17] P. Michaud, A. Seznec, and R. Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA-97)*, June 1997.

[18] Pierre Michaud. A PPM-like, tag-based predictor. *Journal of Instruction Level Parallelism (http://www.jilp.org/vol7)*, April 2005.

[19] Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. Cacti 6.0: A tool to model large caches. Research report hpl-2009-85, HP Laboratories, 2009.

[20] S.T. Pan, K. So, and J.T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1992.

[21] A. Seznec. Analysis of the O-GEHL branch predictor. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, june 2005.

[22] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeidès. Design tradeoffs for the ev8 branch predictor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, 2002.

[23] André Seznec. The L-TAGE branch predictor. *Journal of Instruction Level Parallelism (http://wwwjilp.org/vol9)*, May 2007.

[24] André Seznec. A 64 kbytes ISL-TAGE branch predictor. In *Proceedings of the 3rd Championship Branch Prediction*, June 2011.

[25] André Seznec. Storage Free Confidence Estimation for the TAGE branch predictor. In *Proceedings of the 17th IEEE Symposium on High-Performance Computer Architecture (HPCA 2011)*, Feb 2011.

[26] André Seznec and Pierre Michaud. A case for (partially)-tagged geometric history length predictors. *Journal of Instruction Level Parallelism (http://www.jilp.org/vol8)*, April 2006.

[27] E. Sprangle, R.S. Chappell, M. Alsup, and Y.N. Patt. The agree predictor: A mechanism for reducing negative branch history interference. In 24th *Annual International Symposium on Computer Architecture*, 1995.

[28] T.-Y. Yeh and Y.N. Patt. Two-level adaptive branch prediction. In *Proceedings of the 24th International Symposium on Microarchitecture*, Nov. 1991.