A case for (partially) TAgged GEometric history length branch prediction *

André Seznec SEZNEC@IRISA.FR
Pierre Michaud PMICHAUD@IRISA.FR
IRISA/INRIA/HIPEAC
Campus de Beaulieu, 35042 Rennes Cedex, France

Abstract

It is now widely admitted that in order to provide state-of-the-art accuracy, a conditional branch predictor must combine several predictions. Recent research has shown that an adder tree is a very effective approach for the prediction combination function.

In this paper, we present a more cost effective solution for this prediction combination function for predictors relying on several predictor components indexed with different history lengths. Using GEometric history length as the O-GEHL predictor, the TAGE predictor uses (partially) tagged components as the PPM-like predictor. TAGE relies on (partial) hit-miss detection as the prediction computation function. TAGE provides state-of-the-art prediction accuracy on conditional branches. In particular, at equivalent storage budgets, the TAGE predictor significantly outperforms all the predictors that were presented at the Championship Branch Prediction in december 2004.

The accuracy of the prediction of the targets of indirect branches is a major issue on some applications. We show that the principles of the TAGE predictor can be directly applied to the prediction of indirect branches. The ITTAGE predictor (Indirect Target TAgged GEometric history length) significantly outperforms previous state-of-the-art indirect target branch predictors.

Both TAGE and ITTAGE predictors feature tagged predictor components indexed with distinct history lengths forming a geometric series. They can be associated in a single cost-effective predictor, sharing tables and predictor logic, the COTTAGE predictor (COnditional and indirect Target TAgged GEometric history length).

1 Introduction

State-of-the-art conditional branch predictors [24, 26, 9, 11] exploit several different history lengths to capture correlation from very remote branch outcomes as well as very recent branch history. Hybrid predictors [19] were initially relying on metapredictors to select a prediction from a few different predictions. Several different approaches were later proposed to compute the final prediction from predictors featuring multiple components. E.g., majority vote [21], predictor fusion [18] and partial tagging [6] have been shown to be competitive with meta-prediction.

However, recent studies on conditional branch predictors as illustrated by the 1st Championship Branch Prediction in december 2004 [7, 3, 17, 14, 25] seemed to indicate that the adder tree is the most competitive approach for final prediction computation. Using a (multiply)-adder tree as the prediction combination function was first proposed for the neural based branch predictors [31, 12]. These first proposals were suffering from several

*. This work was partially supported by an Intel research grant and an Intel research equipment donation

©2006 AI Access Foundation and Morgan Kaufmann Publishers. All rights reserved.

shortcomings, for instance high prediction latency (due to the adder tree) and predictor hardware logic complexity (growing linearly with the history length). Most of these shortcomings were progressively addressed. Latency issue was addressed through ahead pipelining [9]; prediction accuracy was improved through different steps: mixing local and global history [13], using redundant history and skewing [23]; hardware logic complexity was reduced through MAC representation [23] or hashing [30]. Finally, the O-GEHL predictor [25, 24] was shown to be able to exploit very long global history lengths in the hundreds bits range. Experiments showed that the O-GEHL predictor achieves state-of-the-art branch prediction accuracy for storage budgets in the 32Kbit-1Mbit range. It uses a medium number N of predictor tables (e.g. 4 to 12) and limited hardware logic for prediction computation (a N-entry 4-bit adder tree). To the best of our knowledge the O-GEHL predictor is the most storage-effective reasonably implementable conditional branch predictor that has been presented so far.

The O-GEHL predictor relies on an adder tree as the final prediction computation function, but its main characteristic is the use of a **geometric series** as the list of the history lengths. This characteristic allows the O-GEHL predictor to exploit very long history lengths as well as to capture correlations on recent branch outcomes. Our first contribution in this paper is to show that partial tagging is a more cost-effective final branch prediction selection function than an adder tree for predictors using a geometric series of history lengths.

We present the TAGE conditional branch predictor. TAGE stands for TAgged GEometric history length. TAGE is derived from Michaud's tagged PPM-like predictor [20]. It relies on a default tagless predictor backed with a plurality of (partially) tagged predictor components indexed using different history lengths for index computation. These history lengths form a **geometric series**. The prediction is provided either by a tag match on a tagged predictor component or by the default predictor. In case of multiple hits, the prediction is provided by the tag matching table with the longest history. Our main contributions on the conditional branch predictor are 1) the use of geometric history length series in PPM-like predictors 2) a new and efficient predictor update algorithm. On the CBP traces, the TAGE predictor outperforms the O-GEHL predictor at equal storage budgets and equivalent predictor complexity (number of tables, computation logic, etc.). Our study points out that the quality of a prediction scheme is very dependent on the choice of the final prediction computation function, but also on the careful design of the predictor update policy. With the proposed update policy, partial tagging therefore appears to be more efficient than adder tree for final prediction computation.

An indirect branch target misprediction and a conditional branch misprediction result in equivalent penalties since both mispredictions can only be resolved at branch execution time. On some applications, for instance server applications, the number of indirect branches is relatively high. On these applications, accurately predicting indirect branch targets is becoming a major issue. The second contribution of this paper is to point out that the structure of the TAGE predictor can be easily adapted to predict indirect branches. We present ITTAGE, an indirect branch target predictor. ITTAGE stands for Indirect branch Target TAgged GEometric history length predictor. ITTAGE relies on the same principles as TAGE, i.e., a default predictor backed by a plurality of (partially) tagged predictor components indexed using global history lengths that form a geometric series. ITTAGE is

shown to be able to exploit the correlation between a branch target and very long global history in the hundred bits range. ITTAGE reaches an unprecedented level of accuracy for an indirect target branch predictor.

Both TAGE and ITTAGE predictors are implemented using tagged predictor components indexed with distinct history lengths forming a geometric series. The COTTAGE, COnditional and indirect Target TAgged GEometric history length, predictor combines a TAGE predictor and a ITTAGE predictor. Indirect targets and conditional branch outcomes are stored in the same tables and part of the predictors logic is shared. COTTAGE is therefore a cost-effective solution for predicting both conditional branches and indirect branch targets.

Paper outline

The remainder of the paper is organized as follows. Section 2 presents our experimental framework for simulation and evaluation of the branch predictors. Section 3 introduces the TAGE predictor principles and evaluates its performance. In Section 4, we present a few last optimizations that can be implemented on the TAGE predictor and evaluate the TAGE predictor in the context of the CBP rules. Section 5 induces the ITTAGE indirect predictor principles. Section 6 points out that the TAGE and ITTAGE predictors can be combined in single cost-effective predictor, the COTTAGE predictor. Finally, Section 7 reviews related work and summarizes this study.

2 Evaluation framework

2.1 Simulation traces and evaluation metric

To allow reproducibility, the simulations illustrating this paper were run using the publicly available traces provided for the 1st Championship Branch Predictor that was held in december 2004 (http://www.jilp.org/cbp/). 20 traces selected from 4 different classes of workloads are used. The 4 workload classes are: server, multi-media, specint, specfp. Each of the branch traces is derived from an instruction trace consisting of 30 million instructions. These traces include system activity.

30 million instruction traces are often considered as short traces for branch prediction studies. However 30 million instructions represent approximately the workload that is executed by a PC under Linux or Windows in 10 ms, i.e., the OS time slice. Moreover, system activity was shown to have an important impact on predictor accuracy [8]. Finally, some traces, particularly server traces, exhibit very large number of static branches. Such traces are not represented in more conventional workloads such as Specint workloads.

The characteristics of the CBP traces are summarized in Table 1. It is noticeable that many traces feature only a few indirect branches, while other traces and particularly server traces feature many indirect branches.

The evaluation metric used in this paper is misprediction per kiloinstructions (misp/KI). Due to space limitations, in many places we will use the average misprediction rate computed as the ratio of the total number of mispredictions on the 20 benchmarks divided by the total number of instructions in the 20 traces.

Since this study was performed on traces, immediate update of the predictor is assumed. On a real hardware processor, the effective update is performed later in the pipeline, at

	FP-1	FP-2	FP-3	FP-4	FP-5	INT-1	INT-2	INT-3	INT-4	INT-5
static branches	444	452	810	556	243	424	1585	989	681	441
dyn. (x10000)	221	179	155	90	242	419	287	377	207	376
static indirect	47	73	102	57	46	58	52	120	60	58
dyn. (x100)	2	3	4	3	1	2	308	5	447	77
	MM-1	MM-2	MM-3	MM-4	MM-5	SER-1	SER-2	SER-3	SER-4	SER-5
static branches	460	2523	1091	2256	4536	10910	10560	16604	16890	13017
dyn. (x10000)	223	381	302	488	256	366	354	381	427	429
static indirect	47	456	137	176	1625	2040	1974	1947	2536	1821
dyn. (x100)	371	866	2018	144	1754	2841	2735	1386	2995	3130

Table 1: Characteristics of the CBP traces

misprediction resolution or at commit time. However, for branch predictors using very long global branch history, the differences of accuracy between a delayed updated predictor and an immediate update predictor are known to be small [10, 26].

2.2 Predictor initialization

Exact reproducibility assumes exact equal initial state. However, branch predictor behaviors might be sensitive to the initialization state of the predictor. Resetting counters before simulating each trace leads to underestimate cold start effects.

In order to approach a realistic initialization point, the simulations presented in this paper assume that the simulations of the 20 traces are chained without resetting the predictor counters. Compared with assuming resetting all the counters before simulating each trace, the discrepancy in prediction accuracy is relatively marginal for the TAGE predictor with moderate storage budget (0.03 misp/KI for a 64Kbits TAGE predictor), but was found to result in larger and significant discrepancies on other predictors.

However, in Section 4, we present the simulation results of the TAGE predictor strictly respecting the 1st Championship Branch Prediction Rules.

2.3 Information used for indexing the branch predictor

For computing the indexes for global history predictors, most studies consider either hashing the conditional branch history with the branch address or hashing the path history with the branch address [22]. Both these solutions lead to consider distinct paths as equal. This phenomenon can be called *path aliasing*. The impact of path aliasing on predictor accuracy is particularly important when a short global history is used.

In order to limit this phenomenon, it was proposed in [24] to include non-conditional branches in the branch history *ghist* (by inserting a taken bit) and to also use a (limited) 16-bit path history *phist* consisting of 1 address bit per branch.

Branch history management The TAGE predictor relies on using a very long global branch history (in the hundred bits range). This global branch history as well as the path history are speculatively updated and must therefore be restored on misprediction. This can be implemented through circular buffers (for instance a 256 bits buffer for the global

history) to store the branch history [15]. Restoring the branch history and the path history consists of restoring the head pointer.

3 The TAGE conditional branch predictor

In this section, we first recall the general principles of using geometric history lengths initially introduced for the O-GEHL predictor [24]. Then we present the TAGE predictor and evaluate its performance.

3.1 GEometric history length prediction

Geometric history length prediction was introduced with the O-GEHL predictor [24]. The predictor features M distinct predictor tables Ti, $0 \leq i < M$ indexed with hash functions of the branch address and the global branch/path history.

Distinct history lengths are used for computing the index of the distinct tables. Table T0 is indexed using the branch address. The history lengths used for computing the indexing functions for tables Ti, $1 \leq i < M$ are of the form $L(i) = \alpha^{i-1} * L(1)$, i.e., the lengths L(i) form a **geometric series**. More precisely, as history lengths are integers, we use $L(i) = (int)(\alpha^{i-1} * L(1) + 0.5)$.

Using a geometric series of history lengths allows to use very long history lengths for indexing some predictor tables, while still dedicating most of the storage space to predictor tables using short global history lengths. As an example on a 8-component predictor, using $\alpha = 2$ and L(1) =2 leads to the following series {0,2,4,8,16,32,64,128}. As pointed out in [24], the exact formula of the series is not important, but the general form of the series is important.

3.2 The TAGE predictor

The TAGE predictor is directly derived from Michaud's PPM-like tag-based branch predictor [20]. 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. These tagged predictor components Ti, $1 \leq i \leq M$ are indexed using different history lengths that form a geometric series, i.e $L(i) = (int)(\alpha^{i-1} * L(1) + 0.5)$.

Throughout this paper, the base predictor will be a simple PC-indexed 2-bit counter bimodal table. An entry in a tagged component consists in a signed counter *ctr* which sign provides the prediction, a (partial) tag and an unsigned useful counter u. Throughout this paper, u is a 2-bit counter and *ctr* is a 3-bit counter.

3.2.1 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.

The overall prediction is computed as in [20]; i.e., the prediction is provided by the hitting tagged predictor component that uses the longest history. Un case of no matching tagged predictor component, the default prediction is used.

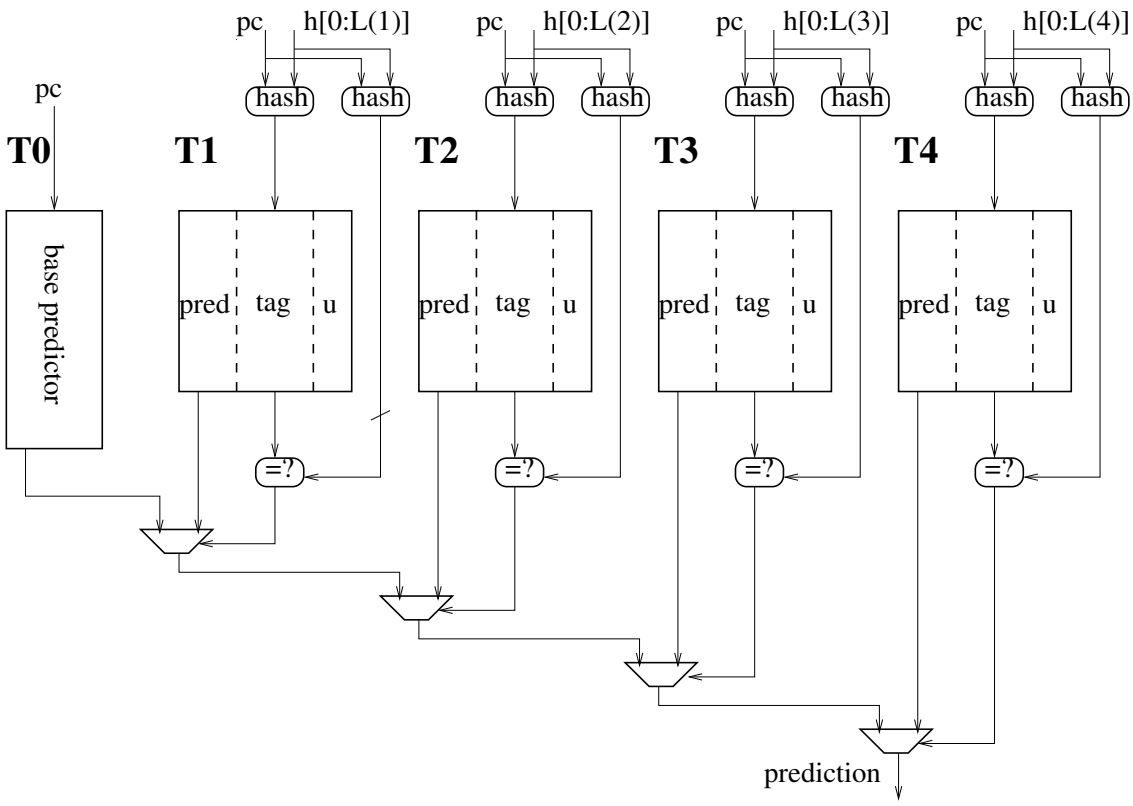

Figure 1: A 5-component TAGE predictor synopsis: a base predictor is backed with several tagged predictor components indexed with increasing history lengths

A few definitions and notations In the remainder of the paper, we define the *provider component* as the predictor component that ultimately provides the prediction. We define the alternate prediction *altpred* as the prediction that would have occurred if there had been a miss on the provider component. That is, if there are tag hits on T2 and T4 and tag misses on T1 and T3, T4 is the provider component and T2 provides *altpred*. If there is no hitting component then *altpred* is the default prediction.

3.2.2 Updating the TAGE predictor

The relatively poor performance observed on the initial PPM-like tag-based predictor is essentially due to the update policy.

We detail here the various scenarios of the update policy that we implement on the TAGE predictor and we contrast this update policy with the original update policy proposed in [20].

Updating the useful counter u The useful counter u of the provider component is updated when the alternate prediction *altpred* is different from the final prediction *pred*. u is incremented when the actual prediction *pred* is correct and decremented otherwise.

Moreover, the useful u counter is also used as an age counter and is gracefully reset as described below. The useful counter is a 2-bit counter. Periodically, the whole column of most significant bits of the u counters is reset to zero, then whole column of least significant bits are reset. In the simulations displayed in this paper, the period for this alternate resetting is 256K branches.

The initial proposition in [20] was missing the graceful resetting of useful counters: some entries were marked useful almost indefinitely.

Update on a correct prediction The prediction counter of the provider component is updated.

The overall prediction is incorrect First we update the provider component prediction counter. As a second step, if the provider component Ti is not the component using the longest history (i.e., $i < M$) , we try to allocate an entry on a predictor component Tk using a longer history than Ti (i.e., $i < k < M$). Note that, unlike in [20], at most a single component is allocated. The allocation process is described below.

The M-i-1 u_j counters are read from predictor components Tj, $i < j < M$. Then we apply the following rules.

(A) Priority for allocation

1) If there exists some k, such that entry $u_k = 0$ then a component Tk is allocated else
2) the u counters from the components Tj, $i < j < M$ are all decremented (no new entry is allocated).

(B) Avoiding ping-phenomenon: If two components Tj and Tk, $j < k$ can be allocated (i.e., $u_j = u_k = 0$) then Tj is chosen with a higher probability than Tk: in the simulations illustrating this paper, the probability to pick Tj is twice the probability to chose Tk. This can be implemented in hardware through using a simple linear feedback register.

(C) Initializing the allocated entry: The allocated entry is initialized with the prediction counter set to weak correct. Counter u is initialized to 0 (i.e., *strong not useful*).

3.2.3 Rationale of the update policy

First of all, the update policy is designed to minimize the perturbation induced by a single occurrence of a branch.

In order to minimize the footprint of a single (branch, history) pair, *at most one* tagged entry is allocated on a misprediction. This has to be contrasted with the original policy in [20] where multiple entries were sometimes allocated.

The selection of the entry to be allocated is managed through the useful counter u. We manage u with two objectives. The value of u can be positive only if there was some positive benefit since the last allocation of the entry: rule (A) guarantees that entries that were recently useful are not reallocated. Second, by decreasing the u counter when the entry is not selected (rules (A.2)) and its graceful reseting, we try to mimic a pseudo Least Recently Useful policy. The useful counter u is set to *strong not useful*: until the entry effectively helps to provide a correct prediction, it is the natural target for the next replacement. This latter property could induce ping-pong phenomena on branches competing for a single entry. Rule (B) was introduced to avoid such ping-pongs.

3.2.4 A simple optimization on newly allocated entries

After the first allocation of a new entry in component C_i, this new entry automatically delivers the prediction on the following occurence of the same (history, program counter) pair.

On some applications, the ratio of mispredictions on these newly allocated entries is very high. For some of the applications (INT-3, MM-1, MM-2), using *altpred* as a prediction is more efficient than the "normal" prediction when the longest history hitting entry is a newly allocated entry. Our experiments showed this property is global to the application and can be dynamically monitored through a single 4-bit counter. Moreover, no special memorization is needed for recording the "newly allocated entry": we consider that an entry is newly allocated if its useful counter is null and its prediction counter is weak (i.e. equal to 0 or -1). This approximation was found to provide results equivalent to effectively recording the "newly allocated entry" information.

3.3 Related work on tagged predictors

The TAGE predictor is directly connected with the PPM predictor from Chen et al.[2]. While PPM predictor was defined to study predictability limits, a hardware implementation of the TAGE predictor seems realistic. The TAGE predictor is also very related to the YAGS predictor [6]. The YAGS predictor features a single level of tagged components.

3.4 Performance evaluation of the TAGE predictor

We will illustrate the accuracy of the TAGE predictor with results for 5-component and 8-component predictors. These numbers of components are chosen because the natural tradeoffs on tag width on the number of entries in default predictor and in the tagged components for these numbers of components lead to storage budgets equal to a power of

Nb of components	storage budget	T0 entries	tag width	u + ctr	tagged Ti entries
5	2^n bits	2^{n-4}	9 bits	2 + 3 bits	2^{n-6}
8	2^n bits	2^{n-4}	11 bits	2 + 3 bits	2^{n-7}

Table 2: Configuration of 5-component and 8-component TAGE predictors

two of bits. Other numbers from 4 to 15 components can naturally be considered. The considered configurations are illustrated in Table 2.

3.4.1 Accuracy of the TAGE predictor

On Figure 2, we illustrate the accuracy of the TAGE predictor against the O-GEHL predictor and the original PPM-like tagged predictor from [20][1]. For all the predictors, the best geometric series of history lengths is illustrated. At equal number of tables, the TAGE predictor outperforms the O-GEHL predictor. For instance, for 8-component predictors, a 64 Kbits O-GEHL predictor achieves 2.83 misp/KI while the corresponding TAGE predictor achieves 2.61 misp/KI. For large predictors, the potential of the TAGE predictor is also higher than the O-GEHL predictor one: 2.05 misp/KI for the 1 Mbit 8-component TAGE predictor against 2.27 misp/KI for the 1 Mbit 8-component O-GEHL predictor. A 128 Kbits 8-component TAGE predictor achieves accuracy in the same range as a 512 Kbits O-GEHL predictor (2.36 misp/KI against 2.34 misp/KI). Note also that, for all predictor sizes, the accuracy advantage of the TAGE predictor over the OGEHL predictor is around 0.2 misp/KI.

The 5-component TAGE predictor also outperforms the 8-component O-GEHL predictor at equal storage size. Its accuracy is constantly lower than the one of the 8-component TAGE component, but by a more limited margin (around 0.1 misp/KI).

Experiments with a 15-component TAGE predictor (not illustrated in Figure 2) essentially showed that increasing the number of components over 8 has no signicant return.

3.4.2 Importance of the update policy

The curve illustrating the original PPM-like predictor (featuring 5 components) on Figure 2 points out the importance of a well engineered update policy to obtain high accuracy. The original PPM-like predictor is essentially a 5-component TAGE predictor, but its accuracy is significantly lower than the one of the 5-component TAGE predictor particularly on small and medium size predictors.

The accuracy discrepancy between the simulation results presented in [20] (3.10 misp/KI) and those in Figure 2 for a 64 Kbits predictor (3.24 misp/KI) is due to the chaining of trace simulations. Considering this chaining enabled us to discover that aging the useful counters was needed.

1. The O-GEHL predictor was shown to significanly outperform the other previously published realistic predictors on the same benchmark set in [24]

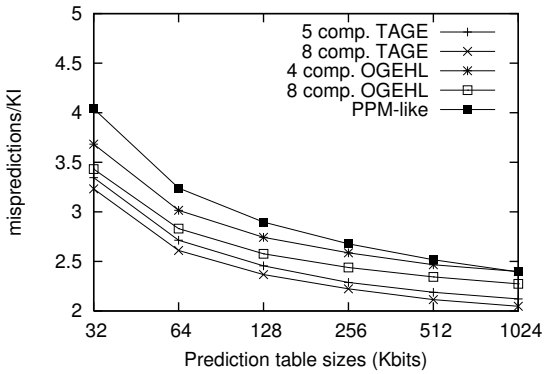

Figure 2: Conditional branch prediction accuracy on the TAGE predictor

The allocation of multiple entries on tagged components in [20] was also a major obstacle of the efficiency: our experiments showed that increasing the number of components over four was resulting in no accuracy benefit while on the TAGE predictor it results in 0.07 to 0.12 misp/KI reduction in average for equivalent storage budget predictors.

The importance of the initialization of the allocated entries was already pointed out in [20] and some hardware was dedicated to control this initialization. But with our improved method, the method used in [20] is clearly suboptimal.

3.4.3 Related works on the importance of update policy

Updating the predictor is not on the critical path. Therefore, complex update policy may be applied. While many studies have detailed update policies in branch predictors, the importance of a careful design of the update policy has been rarely pointed out.

In order to illustrate the importance of the update policy on the effective accuracy of the predictor, we cite three examples in addition of TAGE. The first example is the use of the 2-bit counters in branch prediction [29]. The hysteresis bit averages the predictions on the occurences of the same branch, but it also smoothens aliasing impact. The second example is the use of partial update on multiple component predictors [21]. Such a partial update reduces the impact of aliasing on many predictors. The third example is the use of an update threshold in neural based predictors [12]. Without the use of such an update threshold, the neural based predictors perform very poorly.

3.4.4 Sensivity to history length variation

It was shown in [24] that the O-GEHL predictor is very robust to the choice of the parameters of the geometric series of history lengths.

The same property applies for the TAGE predictor as illustrated for 8-component TAGE predictors featuring from 32K to 1Mbits on Figure 3. On this figure, minimum history L(1) is set to 5, the maximum history length is varied from 50 to 500. For instance, for a 128 Kbits 8-component predictor picking any maximum history length in the range 110-500 leads to approximately the same average predictor accuracy (minimum 2.37 misp/KI, maximum 2.41 misp/KI).

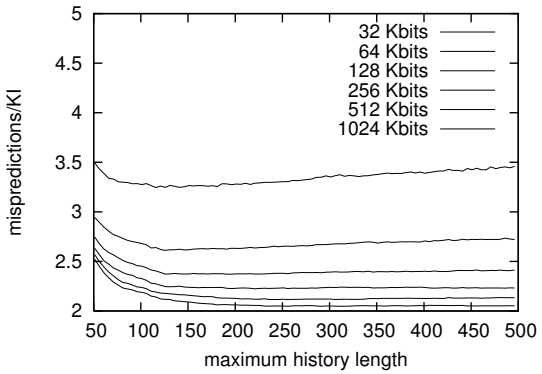

Figure 3: Varying history length parameters

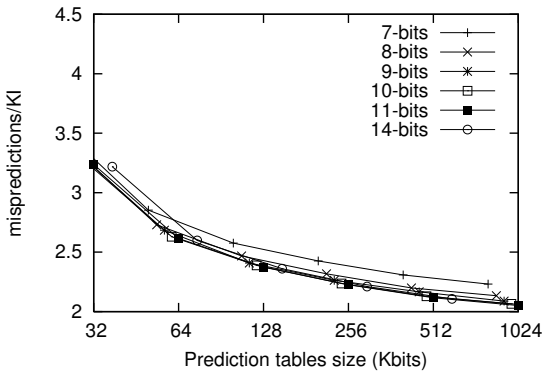

Figure 4: Impact of the tag width on a 8-component TAGE predictor

3.4.5 Tag width tradeoff

Most of the storage budget in the TAGE predictor is dedicated to the storage of the (partial) tags. Therefore one has to correctly dimension the tag width. Using a large tag width leads to waste part of the storage while using a too small tag width leads to false tag match detections. A false tag match detection may result in a misprediction that may trigger a new entry allocation. This new entry allocation may eject some useful prediction, etc.

However, experiments show that increasing the tag width over a threshold varying with the number of components brings little return in accuracy. These thresholds are respectively 9, 11 and 12 for 5, 8 and 14 components TAGE predictors. This is illustrated for a 8-component predictor with varying the threshold from 7 to 14 on Figure 4: using 10 or 11 as tag width appears as a good tradeoff, but even using a 8-bit tag appears to deliver good performance.

3.5 Partial tagging versus adder tree

In this section, we focus on the effectiveness of the two prediction combination functions, adder tree for the O-GEHL predictor and partial tagging for the TAGE predictor.

	FP-1	FP-2	FP-3	FP-4	FP-5
O-GEHL	1.606	0.910	0.430	0.102	0.333
TAGE	1.497	0.795	0.056	0.087	0.025
	INT-1	INT-2	INT-3	INT-4	INT-5
O-GEHL	0.961	6.386	9.949	0.970	0.364
TAGE	0.671	5.562	8.307	0.852	0.301
	MM-1	MM-2	MM-3	MM-4	MM-5
O-GEHL	7.650	9.616	1.043	1.405	5.144
TAGE	7.273	9.374	0.247	1.329	4.430
	SER-1	SER-2	SER-3	SER-4	SER-5
O-GEHL	2.344	2.220	4.912	4.222	3.352
TAGE	1.596	1.106	3.709	2.893	2.226

Table 3: Partial tagging versus adder tree: misp/KI

First, let us point out that the accuracy difference between the 8-component O-GEHL predictor and the 8-component TAGE predictor remains approximately constant (around 0.20 misp/KI) when varying the predictor size from 64 Kbits to 1 Mbit.

Second, the O-GEHL predictor also features dynamic history length fitting [16] that allows it to adapt the used history length to the behavior of the application and this was shown to bring a significant accuracy benefit. Up to now we have not found any elegant and efficient way to implement dynamic history length fitting on the TAGE predictor. Despite the absence of dynamic history length fitting, the TAGE predictor outperforms the O-GEHL predictor.

Therefore, to compare the effectiveness of the two prediction combination functions, we have run simulations of a 8-component 64 Kbits O-GEHL predictor where dynamic history length fitting is disabled (GEHL predictor in [24]). The same geometric series of history lengths are considered for the 64 Kbits GEHL and TAGE predictors, i.e., L(1)=5 and L(7)=130. These parameters are close to the best possible parameters for both predictors. Results of these simulations are displayed on a per benchmark basis on Table 3.

This table clearly illustrates that, despite using 16 bits per entry on a tagged component entry on the TAGE predictor against using only 4 bits per entry on the O-GEHL predictor, partial tagging is more storage effective than the adder tree. On any single benchmark, the TAGE predictor is more accurate than the O-GEHL predictor. It is also noticeable that the advantage of the TAGE predictor is large on the benchmarks with large code footprint, i.e., the server benchmarks. The same experiment was performed for a wide range of (L(1), L(7)) pairs leading to very similar results.

3.6 Useful and Prediction Counter widths

Results displayed in this paper were obtained using 3-bit counters for prediction and 2 bits for the useful counters. On most benchmarks, using 2-bit counters for prediction and a single bit for the useful counters on the tagged tables is sufficient. However, on three benchmarks, INT-3, MM-1 and MM-2, some accuracy difference was encountered.

This translates in an average of 2.78 misp/KI for a 8-component 57K bits TAGE predictor against 2.61 misp/KI for the illustrated 64 Kbits TAGE predictor.

3.7 Associativity on predictor tables

Classicaly on caches, associativity improves miss ratios. On the TAGE predictor, only partial tags are used. At equal partial tag width, two opposite phenomena occur. The number of conflict miss is reduced, but the number of false matches is increased.

Experiments using associativity on predictor tables lead to the following conclusion. Associativity enhances predictor accuracy for small TAGE predictors, particularly on the 32Kbits 5-component TAGE predictor, but is essentially useless for TAGE predictors featuring 64Kbits of storage or more.

As using associativity on a TAGE predictor induces a more complex prediction computation and update logic, its usage is not cost-effective.

3.8 Implementation issues

3.8.1 Prediction response time

The prediction response time on most global history predictors involves three components: the index computation, the predictor table read and the prediction computation logic.

It was shown in [24] that very simple indexing functions using a single stage of 3-entry exclusive-OR gates can be used for indexing the predictor components without impairing the prediction accuracy. In the simulation results presented in this paper, full hash functions were used. However experiments using the 3-entry exclusive-OR indexing functions described in [24] showed a very similar total misprediction numbers (+/- 1%).

The predictor table read delay depends on the size of tables.

On the TAGE predictor, the (partial) tags are needed for the prediction computation. The tag computation may span during the index computation and table read without impacting the overall prediction computation time. Complex hash functions may then be implemented.

The last stage in the prediction computation on the TAGE predictor consists in the tag match followed by the prediction selection. The tag match computations are performed in parallel on the tags flowing out from the tagged components.

At equal storage budgets and equal number of components, the response time of the TAGE predictor will be very close the one of the O-GEHL predictor (equivalent prediction computation time). This response time is slightly longer than the response time of more conventional *gshare* or *2bcgskew* predictors for which the prediction computation is simpler.

3.8.2 Ahead pipelining

In order to provide the prediction in time for next instruction block address generation, ahead pipelining was proposed in [28] and detailed in [27].

The access to the TAGE predictor can be ahead pipelined using the same principle as described for the OGEHL predictor [24]. At cycle T-X, the prediction tables are read using the X-block ahead program counter and the X-block ahead global history. 2^X possible predictions are computed in parallel and information on the last X-block is used to select the final prediction. As on the OGEHL predictor, ahead pipelining induces some loss of

prediction accuracy on medium size predictors, mostly due to aliasing on the base predictor. For instance, on a 64Kbits 8-component predictor, 3-block ahead pipelining reaches 2.73 misp/KI against 2.61 misp/KI on our benchmark set. This small accuracy gap tends to vanish for larger predictors, for instance 2.08 misp/KI against 2.05 misp/KI for 1Mbit 8-component TAGE predictors.

3.8.3 Update implementation

The predictor update must be performed at commit time.

On a correct prediction, only the prediction counter *ctr* and the useful counter u of the matching component must be updated, i.e., a single predictor component is accessed.

On a misprediction, a new entry must be allocated in a tagged component. Therefore, a prediction can potentially induce up to three accesses to the predictor on a misprediction, i.e, read of all predictor tables at prediction time, read of all predictor tables at commit time and write of (at most) two predictor tables at update time. However, the read at commit time can be avoided. A few bits of information available at prediction time (the number of the providing component, *ctr* and u values for the providing component and the nullity of all the u counters) can be checkpointed.

The predictor can therefore be implemented using dual-ported predictor tables. However, most updates on correct predictions concern already saturated counters and can be avoided through checkpointing the information *saturated ctr* and *saturated u*. Since at most two predictor components are updated at commit time, using 2 or 4-bank structure for the predictor tables is a cost-effective alternative to the use of dual-ported predictor tables.

4 Evaluating the TAGE predictor in the context of the CBP rules

The First Championship Branch Prediction was fixing a storage budget of 64Kbits + 256 bits.

Simple tuning for allowing a slightly better usage of the storage budget can be performed on the TAGE predictor. First to enhance the behavior of the bimodal base predictor, one can share an hysteresis counter between several prediction entries as proposed on the EV8 predictor [26]. Using one hysteresis bit for four prediction bits appeared to be a good tradeoff.

Second, we remarked that using slightly different tag widths on the different tagged tables makes a better usage of the storage space. More precisely, the width of the tag should (slightly) increase with the history length: a false match is more harmful on the tables using the longest history.

For a 5-component TAGE predictor, we use respectively 8-bit tags for T1 and T2, and 9-bit tags for T3 and T4. The tagged tables feature 1Kentries, and represent a total of 54 Kbits.

For a 8-component TAGE predictor, we use respectively 9-bit tags for T1 and T2, 10-bit tags for T3 and T4, 11-bit tags for T5 and T6, 12-bit tags for T7. The tagged tables feature 512 entries, and represent a total of 53.5 Kbits.

On both predictors, we use a base bimodal predictor consisting of 8K prediction bits and 2K hysteresis bits, i.e the 5-component and the 8-component CBP-TAGE predictors feature

64Kbits and 63.5 Kbits respectively. The respective history length series are (5,15,44,130) and (5,9,15,25,44,76,130).

The 8-component CBP-TAGE predictor achieves 2.553 misp/KI and the 5-component CBP-TAGE predictor achieves 2.678 misp/KI. This has to be compared with the 2.820 misp/KI achieved by both the 64Kbits CBP O-GEHL predictor and the 64Kbits Gao and Zhou predictor [7].

Benchmark by benchmark simulation results are displayed in Table 4.

	FP-1	FP-2	FP-3	FP-4	FP-5
8C-TAGE	1.489	0.787	0.061	0.092	0.028
5C-TAGE	1.609	0.814	0.060	0.091	0.028
Gao[7]	1.156	0.924	0.019	0.031	0.012
OGEHL	1.408	0.906	0.413	0.181	0.041
	INT-1	INT-2	INT-3	INT-4	INT-5
8C-TAGE	0.631	5.482	8.203	0.818	0.307
5C-TAGE	1.228	5.864	8.836	0.954	0.302
Gao[7]	0.964	6.403	7.063	1.366	0.320
OGEHL	0.694	5.519	8.998	0.940	0.343
	MM-1	MM-2	MM-3	MM-4	MM-5
8C-TAGE	7.242	9.373	0.244	1.319	4.306
5C-TAGE	7.335	9.446	0.240	1.280	4.464
Gao[7]	6.953	8.974	0.305	1.209	4.780
OGEHL	7.218	9.019	0.229	1.358	4.427
	SER-1	SER-2	SER-3	SER-4	SER-5
8C-TAGE	1.259	1.191	3.452	2.629	2.157
5C-TAGE	1.313	1.249	3.511	2.697	2.241
Gao[7]	2.154	2.190	4.691	3.728	3.209
OGEHL	1.999	1.912	4.422	3.407	2.956

Table 4: Accuracy of the 64Kbits TAGE predictor (misp/KI) using CBP contest rules

5 The ITTAGE indirect jump target predictor

Some applications feature a quite significant proportion of indirect jumps. Considering only the 5 server traces in our benchmark set, a total of 1309 K indirect jumps, i.e., 8.9 indirect jumps per 1000 instructions, are encountered while on the same set of benchmarks, a 64 Kbits 8-component TAGE predictor encounters only 337K conditional branch mispredictions representing an average of 2.15 misp/KI. As an indirect jump target misprediction and a conditional branch mispredicton result in equivalent mispenalties, specific indirect jump target predictors are needed.

Gshare-like indexed jump predictors were first proposed by Chang et al [1]. This proposal was refined by Driesen and Holzle [4]. The cascaded indirect branch predictor proposed

in [4] is representative of current state-of-the-art indirect branch predictors. It features a PC indexed first-level predictor backed with a second-level tagged table indexed with a hash function of the global history and the PC.

In this section, we introduce the ITTAGE indirect target predictor based on the same principles as the TAGE predictor. The ITTAGE predictor outperforms the cascaded indirect branch predictor at equivalent storage budget.

Remark On architectures featuring a BTB, the BTB can provide the function of the base indirect jump predictor. For the sake of simplicity of the presentation, we will not consider this case in this paper. However adapting our ITTAGE predictor to an architecture featuring a BTB is straightforward.

5.1 ITTAGE predictor principles

Applying the principles of the TAGE predictor to indirect target prediction is straightforward. The Indirect Target TAgged GEometric length, ITTAGE, predictor features a tagless base predictor IT0 in charge of providing a default prediction and a set of (partially) tagged predictor components. This set of tagged predictor components ITi, $1 \leq i < M$ is indexed using different history lengths that form a geometric series, i.e $L(i) = (int)(\alpha^{i-1} * L(1) + 0.5)$. This is illustrated in Figure 5. The counters representing predictions in TAGE are replaced by full target address *Targ* plus a confidence bit c to get some hysteresis on the predictor.

The prediction selection algorithm is directly inherited from the TAGE predictor. The predictor update algorithm is also inherited from the TAGE update algorithm apart that target replacement on the base predictor and on the tagged components in case of a hit but false prediction is controlled through a single confidence bit.

Related work on indirect jump target predictions

Relatively few studies have considered jump target predictions. The cascaded indirect jump predictor [4] uses a PC indexed table and gshare-like indexed predictor table. The PC indexed table is used to filter indirect jumps that always have the same target. The cascaded indirect jump predictor can be considered as a 2-component ITTAGE predictor. Driesen and Holzen [5] proposed a generalized form of the cascaded predictor, the multistage cascaded predictor, but limited their study to very short history length, 12 bits.

5.2 Performance evaluation of the ITTAGE predictor

We illustrate the accuracy of the ITTAGE predictor on our reference benchmark set for ITTAGE predictors with respectively 5 and 8 components. As on the TAGE predictor, more entries are required on the default tagless component than on the other components. We found that using approximately the same total number of tagged entries and non-tagged entries results in a good trade-off, i.e., using 2^n entries on IT0 and 2^{n-2} (resp. 2^{n-3}) entries on the tagged components for the 5-component (resp. 8-component) ITTAGE predictor.

As CBP traces are IA32 traces, 32 bit addresses must be predicted. Entries on the tagless tables feature 33 bits (jump target +confidence bit)

An entry on a tagged tabble consists of the partial tag, the jump target, the confidence bit and a 2-bit useful counter. Using 9 and 11 bits tag respectively width were found to be good tradeoffs for 5 and 8 components respectively. Therefore entries on the tagged tables feature 44 and 46 bits respectively for 5 and 8 components respectively.

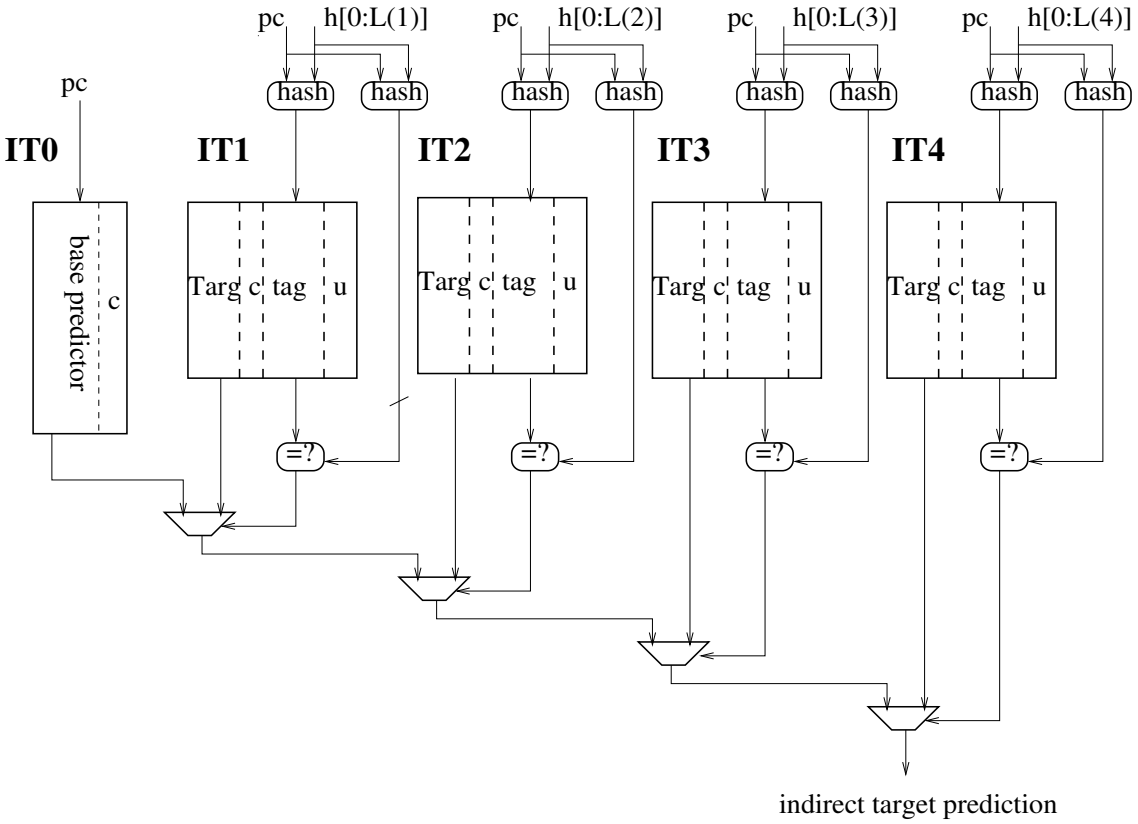

Figure 5: ITTAGE predictor structure: a base predictor is backed with several tagged predictor components indexed with increasing history lengths

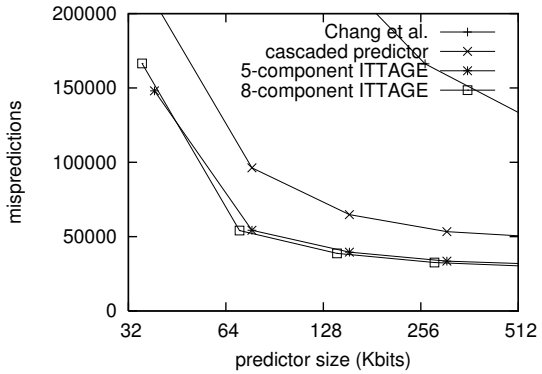

Figure 6: Mispredictions on indirect jump predictors

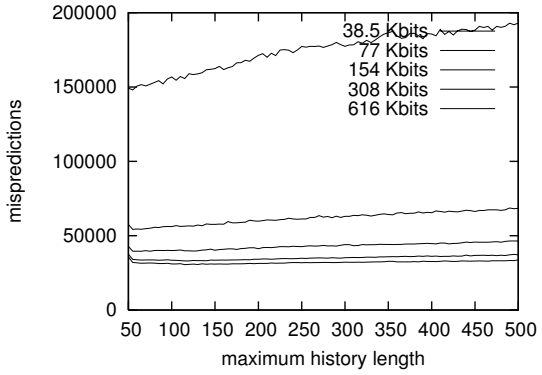

Figure 7: Varying the maximum history length on a 5-component ITTAGE predictor

The storage budgets for the simulated predictors are therefore 77 $*2^n$ bits for the 5-component ITTAGE predictor and 72.25 $*2^n$ bits for the 8-component ITTAGE predictor We also simulated the gshare-like indexed single table indirect jump predictor from Chang et al [1] and the 2-table cascaded predictor [4]. For this latter predictor, on the CBP benchmark using the same number of entries are considered on the two tables,a 1-bit confidence is used on both levels, 9 tag bits are used on the second level since this width is sufficient in practice. For each of the simulated predictors, we illustrate the best configuration (i.e., the best history length series).

Figure 6 shows that the ITTAGE predictor allows to limit the number of mispredictions on indirect jumps. For instance, using a 144.5 Kbits 8-component ITTAGE results in less than 39,000 mispredictions, i.e., a 2 % misprediction rate against 65,000 mispredictions i.e., a 3.4 % misprediction rate when using a 154Kbits cascaded predictor.

As expected from the experiments on the TAGE and O-GEHL predictors, sensitivity to variations of the parameters of the geometric series is very low for the ITTAGE predictor. Figure 7 illustrates this phenomenon on a 5-component ITTAGE predictor.

6 COTTAGE: putting the ITTAGE predictor and the TAGE predictor together

The ITTAGE predictor is effective to predict indirect jump targets. While some class of applications would benefit from the use of such a state-of art indirect target predictor, many other applications do not feature significant numbers of indirect branches (see Table 1). Therefore the cost-effectiveness of implementing an ITTAGE predictor might be questionable as it features several extra tables compared with previous indirect jump predictor proposals: using an 72.25 Kbits 8-component predictor leads to 0.09 misp/KI in average on the 20 benchmarks against 0.16 misp/KI using a 77 Kbits 2-component cascaded predictor and 0.39 misp/KI for a 132 Kbits single table indirect jump predictor. While the extra cost of a second table in the indirect branch predictor is compensated by a significant overall misprediction reduction, using 3 or 6 extra tables (and the associated logic for index/tag computation and tag check) for implementing the ITTAGE predictor does not seem cost-effective.

On high-end superscalar processors, all the prediction structures (conditional branch predictors, indirect jump predictors, return stack, BTB) must be accessed speculatively. The similarities of the structures of TAGE predictor and the ITTAGE predictor allow to regroup the conditional branch predictor and the indirect jump predictor in a single physical structure as illustrated in Figure 8. The COTTAGE predictor (for COnditional and Target TAgged GEometric history length) implements both a TAGE predictor and an ITTAGE predictor. The COTTAGE predictor components store both conditional and indirect jump predictions. Tag and index computation logic is also shared. On each cycle, conditional branch prediction and indirect jump prediction are read on the same row.

For cost-effectiveness, the predictor tables must implement more TAGE entries than ITTAGE entries. Therefore, a row in a predictor table stores a single indirect jump prediction and several conditional branch predictions. For simplifying Figure 8, the multiplexors needed to select the correct TAGE entry at each COTTAGE component exit have been omitted.

The robustness of both TAGE and ITTAGE predictors to the choice of the geometric series of history length parameters allows high accuracy on both predictors (see Figures 3 and 7). For instance when associating a 8-component ITTAGE predictor featuring 72.25 Kbits predictor with a 256Kbits TAGE predictor, if quasi-optimal parameters for the TAGE predictor are used such as L(1)=5 and L(7)=195, the ITTAGE predictor encounters 58614 mispredictions against 54103 mispredictions for its best parameters, i.e., L(1)=5 and L(7)=66.

The COTTAGE predictor appears as a cost-effective solution when one considers the cost of implementing both a conditional predictor and an indirect efficient indirect jump predictor: a 5-component COTTAGE predictor provides state-of-the-art prediction accuracy on both conditional branches and indirect jumps while using a total of only 5 storage tables.

7 Conclusion

It is now widely admitted that conditional branch predictors must exploit several different history lengths to capture correlation from very remote branch outcomes as well as very

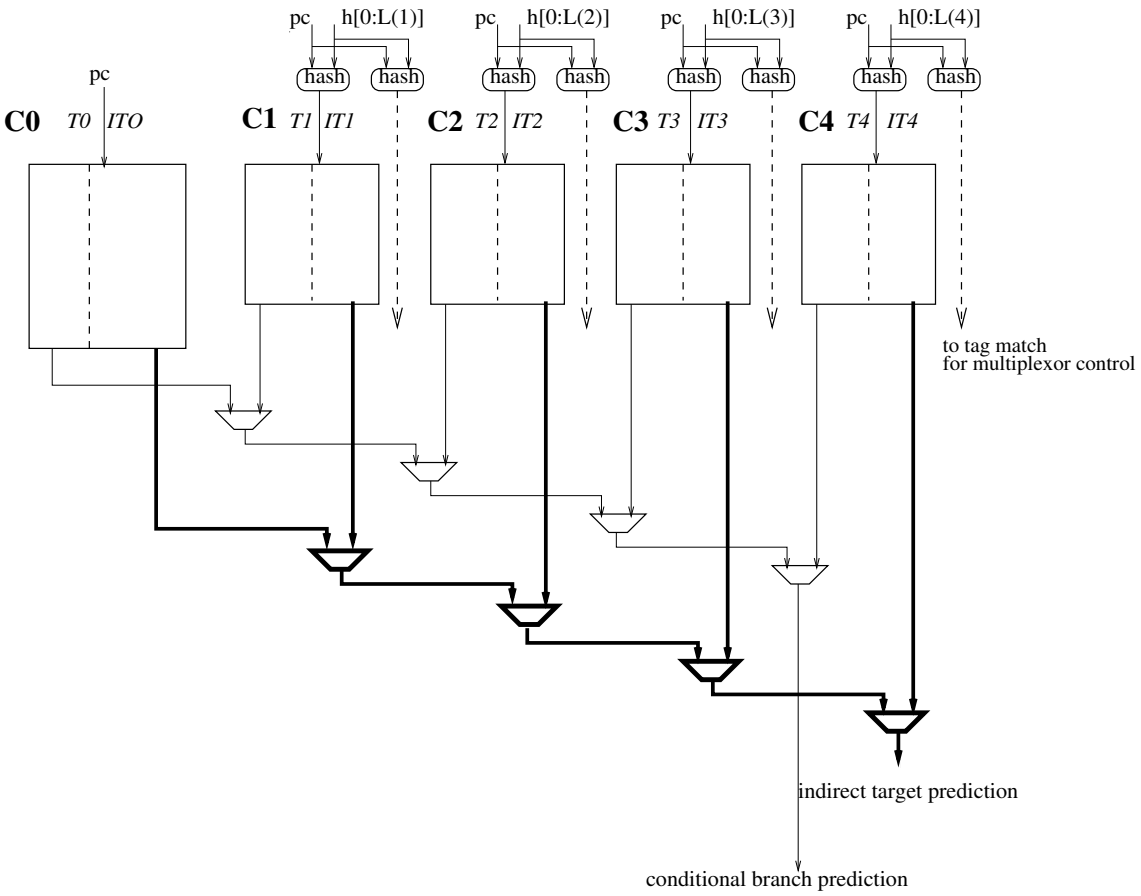

Figure 8: The COTTAGE predictor: combining a ITTAGE predictor and a TAGE predictor

recent branch history [24, 26, 9]. Before this study, recent research [9, 24, 30] on branch prediction seemed to indicate that the most cost-effective way to combine these predictions was through the (multiply)-accumulation of the predictions. Among these new proposals, the O-GEHL predictor was shown to represent the state-of-the-art at the 1st Championship Branch Prediction in december 2004 [24].

We have shown that, for predictors using geometric series of history lengths (as O-GEHL), using partial tags is more storage effective for achieving high accuracy. At equal number of predictor tables, the TAGE predictor outperforms the O-GEHL predictor with lower hardware complexity (no use of dynamic history length fitting, shorter global history).

Moreover while combining predictions through (multiply)-accumulation is clearly limited to conditional branch prediction, a very efficient indirect branch predictor, the ITTAGE predictor can be designed using the same principles as the TAGE predictor.

Finally, we have presented the COTTAGE predictor, a cost effective solution for predicting both conditional and indirect branches. A M-component COTTAGE predictor embeds both a M-component TAGE predictor and a M-component ITTAGE predictor, but features only M predictor tables. Moreover some of the logic (index and tag computation) is also shared between the TAGE and ITTAGE predictors.

The TAGE and ITTAGE predictors inherit most of the properties of the O-GEHL predictors. The hardware complexity of the computation logic is limited and scales only linearly with the number of components in the predictor. The design space of cost-effective COTTAGE predictors is large. For instance, high level of accuracy are obtained for a broad spectrum of maximum history lengths ranging from 100 to 500 for medium size predictors.

Moreover as the other global branch history predictors [27, 9, 23, 30, 24], the COTTAGE predictor can be ahead pipelined to provide predictions in time for the instruction fetch engine applying the general technique that was described in [24].

This study has demonstrated the superiority of using partial tagging as a prediction combination function over using an adder tree for predictors using multiple global history indexed components. However, if a predictor also includes components using other information sources (local history, loop counts, value prediction, ..), an other final prediction combination function has to be found.

Simulator distribution

The simulator of the CBP TAGE predictor described in Section 4 is accessible on the website of the authors at http://www.irisa.fr/caps/.

References

[1] P.-Y. Chang, E. Hao, and Y. N. Patt. Target prediction for indirect jumps. In *ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture*, pages 274–283, New York, NY, USA, 1997. ACM Press.

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

[3] V. Desmet, H. Vandierendonck, and K. D. Bosschere. 2far: A 2bcgskew predictor fused by an alloyed redundant history skewed perceptron branch predictor. *Journal of Instruction Level Parallelism (http://www.jilp.org/vol7)*, 2005.

[4] K. Driesen and U. Holzle. The cascaded predictor: Economical and adaptive branch target prediction. In *Proceeding of the 30th Symposium on Microarchitecture*, Dec. 1998.

[5] K. Driesen and U. Hölzle. Multi-stage cascaded prediction. In *Proceedings of the 5th International Euro-Par Conference on Parallel Processing, EURO-PAR'99*, volume 1685 of *LNCS*, pages 1312–1321. Springer-Verlag, 1999.

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

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

[8] N. Gloy, C. Young, J. B. Chen, and M. D. Smith. An analysis of dynamic branch prediction schemes on system workloads. In *ISCA '96: Proceedings of the 23rd annual international symposium on Computer architecture*, pages 12–21. ACM Press, 1996.

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

[13] D. Jiménez and C. Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems*, November 2002.

[14] D. A. Jiménez. Idealized piecewise linear branch prediction. *Journal of Instruction Level Parallelism (http://www.jilp.org/vol7)*, April 2005.

[15] S. Jourdan, T.-H. Hsing, J. Stark, and Y. N. Patt. The effects of mispredicted-path execution on branch prediction structures. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, October 1996.

[16] T. Juan, S. Sanjeevan, and J. J. Navarro. A third level of adaptivity for branch prediction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 30 1998.

[17] G. Loh. Deconstructing the frankenpredictor for implementable branch predictors. *Journal of Instruction Level Parallelism (http://www.jilp.org/vol7)*, April 2005.

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

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

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

[21] 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.

[22] R. Nair. Dynamic path-based branch prediction. In *International Symposium on Microarchitecture (MICRO-28)*, pages 15–23, 1995.

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

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

[25] A. Seznec. Genesis of the o-gehl branch predictor. *Journal of Instruction Level Parallelism (http://www.jilp.org/vol7)*, April 2005.

[26] 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.

[27] A. Seznec and A. Fraboulet. Effective ahead pipelining of the instruction addres generator. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.

[28] A. Seznec, S. Jourdan, P. Sainrat, and P. Michaud. Multiple-block ahead branch predictors. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 116–127, 1996.

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

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

[31] L. N. Vintan and M. Iridon. Towards a high performance neural branch predictor. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings.*, 1999.