

A 256 Kbits L-TAGE branch predictor *

André Seznec
IRISA/INRIA/HIPEAC

Abstract

The TAGE predictor, TAgged GEometric length predictor, was introduced in [10].

TAGE relies on several predictor tables indexed through independent functions of the global branch/path history and the branch address. The TAGE predictor uses (partially) tagged components as the PPM-like predictor [5]. It relies on (partial) match as the prediction computation function. TAGE also uses GEometric history length as the O-GEHL predictor [6], i.e., the set of used global history lengths forms a geometric series, i.e., $L(j) = \alpha^{j-1}L(1)$. This allows to efficiently capture correlation on recent branch outcomes as well as on very old branches.

For the realistic track of CBP-2, we present a L-TAGE predictor consisting of a 13-component TAGE predictor combined with a 256-entry loop predictor. This predictor achieves 3.314 misp/KI on the set of distributed traces.

Presentation outline

We first recall the TAGE predictor principles [10] and its main characteristics. Then, we describe the L-TAGE configuration submitted to CBP-2 combining a loop predictor and a TAGE predictor. Section 3 discusses implementation issues on the L-TAGE predictor. Section 4 presents simulation results for the submitted L-TAGE predictor and a few other TAGE predictor configurations. Section 5 briefly reviews the related works that had major influences in the L-TAGE predictor proposition and discusses a few tradeoffs that might influence the choice of a TAGE configuration for an effective implementation.

1. The TAGE conditional branch predictor

The TAGE predictor is derived from Michaud's PPM-like tag-based branch predictor [5] and uses geometric history lengths [6]. 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 T_i . These tagged predictor components T_i , $1 \leq i \leq M$ 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; in order to save storage space, the hysteresis bit is shared among several counters as in [7].

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.

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 hitting component then *altpred* is the default prediction.

1.1. 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.

However, we found that, on several applications, using the alternate prediction for newly allocated entries is more efficient. Our experiments showed this property is essentially global to the application and can be dynamically monitored through a single 4-bit counter (*USE_ALT_ON_NA* in the simulator). On the predictor an entry is classified as "newly allocated" if its prediction counter is weak.

Therefore 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 predic-

* This work was partially supported by an Intel research grant, an Intel research equipment donation and by the European Commission in the context of the SARC integrated project #27648 (FP6).

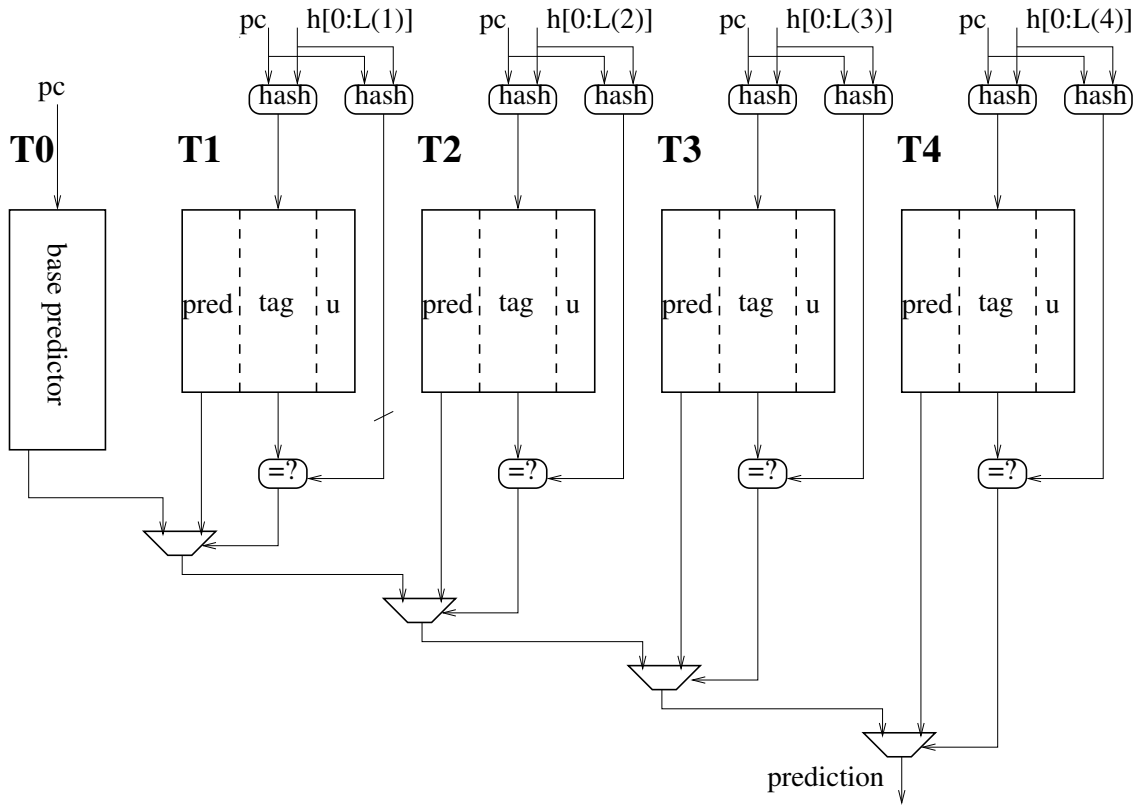


Figure 1. A 5-component TAGE predictor logical synopsis: a base predictor is backed with several tagged predictor components indexed with increasing history lengths. On an effective implementation, predictor selection would be performed through a tree of multiplexors

tion counter sign provides the prediction else the prediction is the alternate prediction

1.2. Updating the TAGE predictor

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 $pred$.

The useful u counter is also used as an age counter and is gracefully reset as described below. We use an aging algorithm resetting alternatively the bits of the u counters.

As a small improvement on the updating presented in [10], after the reset, we flip the signification of the bits u_0 and u_1 of the useful counter till the next reset:

- reset No $2n-1$: $u_1=0$; until reset No $2n$: $u = 2u_1+u_0$
- reset No $2n$: $u_0=0$; until reset No $2n+1$: $u = 2u_0+u_1$
- reset No $2n+1$: $u_1=0$; until reset No $2n+2$: $u = 2u_1+u_0$

The period used in the presented predictor for this alternate resetting is 512K branches.

Updating the prediction counters The prediction counter of the provider component is updated. When the useful counter of the provider component is null, the alternate prediction is also updated.

Allocating tagged entries on mispredictions On mispredictions at most one entry is allocated.

If the provider component T_i is not the component using the longest history (i.e., $i \leq M$), we try to allocate an entry on a predictor component T_k with $i < k \leq M$

The allocation process is described below.

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

- (A) Avoiding ping-pong phenomenon: in the presented predictor, the search for a free entry begins on table T_b , with $b=i+1$ with probability $1/2$, $b=i+2$, with probability $1/4$ and $b=i+3$ with probability $1/4$.

The pseudo-random generator used in the presented predictor is a simple 2-bit counter.

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

2. Characteristics of the submitted L-TAGE predictor

2.1. Information used for indexing the branch predictor

2.1.1. Path and branch history The predictor components are indexed using a hash function of the program counter, the global branch history *ghist* (including non-conditional branches as in [6]) and a (limited) 16-bit path history *phist* consisting of 1 address bit per branch.

2.1.2. Discriminating kernel and user branches Kernel and user codes appear in the traces. In practice in the traces, we were able to discriminate user code from kernel through the address range. In order to avoid history pollution by kernel code, we use two sets of histories: the user history is updated only on user branches, kernel history is updated on all branches.

2.2. Tag width tradeoff

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. Experiments showed that one can use narrower tags on the tables with smaller history lengths.

2.3. Number of the TAGE predictor components

For a 256 Kbits predictor, the best accuracy we found is achieved by a 13 components TAGE predictor.

2.4. The submitted L-TAGE predictor

2.4.1. The loop predictor component The loop predictor simply tries to identify regular loops with constant number of iterations.

The loop predictor provides the global prediction when the loop has successively been executed 3 times with the same number of iterations. The loop predictor used in the submission features 256 entries and is 4-way associative.

Each entry consists of a past iteration count on 14 bits, a current iteration count on 14 bits, a partial tag on 14 bits, a confidence counter on 2 bits and an age counter on 8 bits, i.e. 52 bits per entry. The loop predictor storage is therefore 13 Kbits.

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 255. Age is decremented whenever the entry was

a possible replacement target and incremented when the entry is used and has provided a valid prediction. Age is reset to zero whenever the branch is determined as not being a regular loop.

2.4.2. The TAGE predictor component The TAGE predictor features 12 tagged components and a base bimodal predictor. Hysteresis bits are shared on the base predictor. Each entry in predictor table T_i features a W_i bits wide tag, a 3-bit prediction counter and a 2-bit useful counter.

The submitted predictor uses 4 as its minimum history length and 640 as its maximum history length.

The characteristics of the TAGE component are summarized in Table 1. The TAGE predictor features a total of 241.5 Kbits of prediction storage.

2.4.3. Total predictor storage budget Apart the prediction table storage, the predictor uses two 640 bits global history vectors, two 16 bits path history vectors, a 4 bits USE_ALT_ON_NA counter, a 19 bits counter for gracefully resetting the u counters, a 2-bit counter as pseudo-random generator and a 7-bit counter WITHLOOP to determine the usefulness of the loop predictor. That is an extra storage of 1344 bits.

Therefore the predictor uses a total of $(241.5+13)*1024 + 1344 = 261,952$ storage bits.

3. Implementation issues

3.1. The prediction response time

Since the loop predictor features a small number of entries, the response time of the submitted predictor is dominated by the TAGE 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 [6] that very simple indexing functions using a single stage of 3-entry exclusive-OR gates can be used for indexing the predictor components without significantly 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 [6] showed a very similar total misprediction numbers (+0.03 misp/KI).

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

	Base	T1,T2	T3,T4	T5	T6	T7	T8,T9	T10	T11	T12
history length		4,6	10,16	25	40	64	101,160	254	403	640
Nb entries	16K pred. 4K hyst.	1K	2K	2K	2K	1K	1K	1K	0.5K	0.5K
Tag width		7	8	9	10	11	12	13	14	15
storage budget (bits)	20K	12K	26K	28K	30K	16K	17K	18K	9.5K	10K

Table 1. Characteristics of the TAGE predictor components

prediction selection. The tag match computations are performed in parallel on the tags flowing out from the tagged components.

Therefore, on an aggressively pipelined processor, the response time of the L-TAGE predictor is unlikely to be a single cycle, but may be close to three cycles.

Therefore if the submitted L-TAGE predictor was to be implemented directly, it should be used as an overriding predictor associated with a fast predictor (e.g. a bimodal table).

3.2. How to address TAGE predictor response time: ahead pipelining

In order to provide the prediction in time for next instruction block address generation, ahead pipelining was proposed in [9] and detailed in [8] for global history/path branch predictors. Therefore the access to the TAGE predictor can be ahead pipelined using the same principle as described for the OGEHL predictor [6] and illustrated on Figure 2.

The prediction tables are read using the X-branch ahead program counter, the X-branch ahead global history and the X-branch ahead path history. On each of the tables, 2^{X-1} adjacent entries are read. 2^{X-1} possible predictions are computed in parallel and information on the last X-1 branches (1 bit per intermediate branch) is used to select the final prediction. Ahead pipelining induces some loss of prediction accuracy on medium size predictors, mostly due to aliasing on the base predictor. It is also necessary to checkpoint 2^{X-1} predictions to be able to resume without delay on a branch misprediction [9].

For transitions from user mode to kernel mode and vice-versa, we make the following hypothesis: 1) The first two branches after a trap or an exception are predicted in a special way using 1-block ahead information for the first one and 2-block ahead information for the second one. 2) The first two branches after the return in the user code are predicted with the ahead information available before the exception or the trap.

3.3. Other implementation issues

3.3.1. Number of predictor components The complexity of a design, its silicon area and its power consumption

increase with the number of components. For a 256 Kbits predictor, the best accuracy we found is achieved by a 14-component L-TAGE predictor. However, the TAGE predictor is also quite efficient with a more limited number of components [10].

3.3.2. Predictor update implementation issues The predictor update is performed after commit. Therefore the update logic is not on the critical path.

On a correct prediction, at most two prediction counters *ctr* and the useful counter *u* of the matching component must be updated, i.e., at most two predictor components are accessed.

On a misprediction, a new entry is allocated on 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 numbers of the provider component, the alternate component, *ctr* and *u* values for these two components and the nullity of all the *u* counters) can be checkpointed.

The predictor can therefore be implemented using dual-ported predictor components. However, most updates on correct predictions concern already saturated counters and can be avoided through checkpointing the information *saturated ctr* and *saturated u*. Using 2 or 4-bank structure for the predictor tables (as on EV8 predictor [7]) is a cost-effective alternative to the use of dual-ported predictor tables.

3.3.3. Speculative management

Simple speculative history management On predictor relying only on global history and/or path history such as TAGE, the speculative management of histories can be implemented through circular buffers [3]. Restoring the branch history (respectively the path history) consists of restoring the head pointer.

But complex iteration count management In a deeply pipelined processor, the effectivity of a loop predictor depends on an accurate management of the speculative iteration counts since several iterations of the same loop can

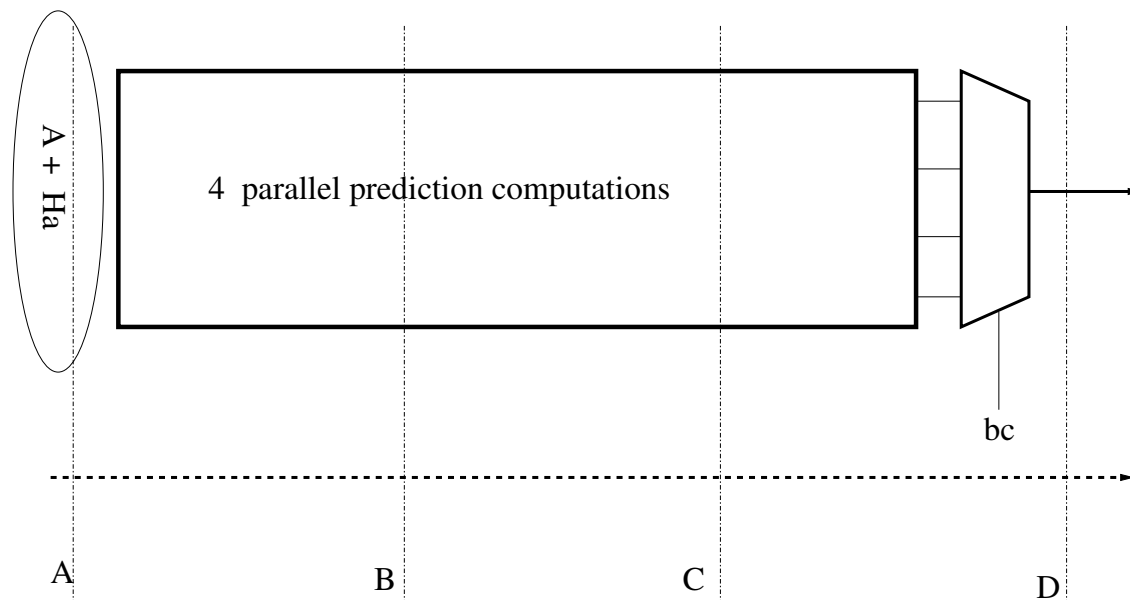


Figure 2. Principle of 3-block ahead branch prediction: information on branch A is used to predict the output of branch C; information on block B and C is used to select the final prediction

be in flight at the same time. A loop predictor is implemented used on the Pentium-M. Therefore this complexity should be considered as manageable.

4. Predictor accuracy

Results per application on the distributed set of traces are displayed in Table 2 for the submitted L-TAGE predictor.

In order to illustrate the potential of the TAGE predictor, we also present simulation results for TAGE predictors featuring simpler hardware complexity: the included 241,5 Kbits TAGE component, a 256 Kbits 13-component TAGE predictor (13C) and a 256 Kbits 8-component TAGE predictor (8C). Finally, as the TAGE predictor can deliver prediction in time provided that ahead pipelining is used, we also illustrate simulations results for a 3-branch ahead 256 Kbits TAGE predictor (8C-Ahead).

The average accuracy of the submitted predictor is **3.314 misp/KI** on the distributed set of traces. When the loop predictor is turned off, the 241,5 Kbits TAGE component achieves 3.368 misp/KI. When the total of the 256 Kbits are affected to the 13 components of TAGE, 3.357 misp/KI is achieved. A 256 Kbits 8-component TAGE predictor achieves 3.446 misp/KI while a 256 Kbits 3-branch ahead TAGE predictor achieves 3.552 misp/KI.

It can be noted that the benefit of the loop predictor is essentially marginal apart on 164.gzip. Simulations on other

sets of traces confirm that, only very rare applications effectively benefit from the loop predictor, therefore associating the loop predictor with TAGE is probably not worth the complexity for a real implementation.

Using a medium number of components (8) in TAGE instead of the best number of components (13) impacts the accuracy only slightly: for the final designer the choice of the number of components will be a tradeoff between the extra complexity induced by using more predictor tables and a small accuracy loss. Finally, ahead pipelining does not impair very significantly the predictor accuracy and can therefore be considered for delivering the prediction in time.

5. Conclusion

The use of multiple global history lengths in a single branch predictor was initially introduced in [4], then it was refined by Evers et al. [2] and further appeared in many proposals. Using tagged predictors was suggested for the PPM predictor from Chen et al.[1]. A first PPM-like implementable version was proposed in [5]. TAGE enhances this first proposition by an improved update policy. The TAGE predictor directly inherits the use of geometric history length series from the OGEHL predictor [6], but is more storage-effective. Using only a limited storage, the loop predictor allows to capture some behaviors that are not captured by the TAGE predictor.

The submitted L-TAGE predictor can be directly adapted to hardware implementation as a multi-cycle overriding pre-

	164	175	176	181	186	197	201	202	205	209
L-TAGE	10.074	9.010	3.222	9.049	2.442	5.152	5.712	0.371	0.346	2.339
241.5 Kbits TAGE	10.789	9.015	3.263	9.055	2.445	5.156	5.868	0.372	0.352	2.347
13C	10.781	8.990	3.224	9.006	2.415	5.141	5.853	0.368	0.349	2.345
8C	10.615	9.080	3.369	9.446	2.534	5.352	5.914	0.379	0.496	2.368
8C-Ahead	10.854	9.384	3.627	9.688	2.728	5.438	6.024	0.406	0.515	2.439
	213	222	227	228	252	253	254	255	256	300
L-TAGE	1.080	1.068	0.386	0.592	0.219	0.311	1.460	0.139	0.036	13.284
241.5Kbits TAGE	1.121	1.110	0.399	0.594	0.219	0.325	1.464	0.141	0.041	13.288
13C	1.119	1.114	0.397	0.590	0.218	0.325	1.547	0.141	0.041	13.269
8C	1.147	1.146	0.542	0.633	0.240	0.378	1.513	0.145	0.041	13.528
8C-Ahead	1.195	1.188	0.571	0.690	0.246	0.398	1.581	0.169	0.043	13.868

Table 2. Per benchmark accuracy in misp/KI

dictor backing a fast single-cycle predictor (e.g. a bimodal predictor) and achieves very high accuracy.

This configuration was submitted because it achieves very accuracy while it could be implemented in hardware. However for an effective hardware implementation, the hardware complexity of the submitted L-TAGE predictor should be compared with other TAGE-based predictor solutions evaluated in Section 4. These solutions might more cost-effective tradeoffs between hardware complexity and predictor performance; in particular:

- The complexity of a real hardware loop predictor is higher than only reflected by its prediction storage budget; the management of the speculative iteration counts might be a major source of hardware logic complexity. The small accuracy benefit brought by the loop predictor is probably not worth this extra complexity.
- The number of components in the submitted predictor is high: using a smaller number of components, e.g. 8, might be a better design tradeoff.
- The L-TAGE predictor would have a multicycle response time: using a slightly less accurate but ahead pipelined TAGE predictor might allow the design of an overall more efficient instruction fetch front-end [8].

References

- [1] I.-C.K. Chen, J.T. Coffey, and T.N. Mudge. Analysis of branch prediction via data compression. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1996.
- [2] 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 *23rd Annual International Symposium on Computer Architecture*, pages 3–11, 1996.
- [3] Stephan Jourdan, Tse-Hao Hsing, Jared Stark, and Yale 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.
- [4] S. McFarling. Combining branch predictors. TN 36, DEC WRL, June 1993.
- [5] Pierre Michaud. A ppm-like, tag-based predictor. *Journal of Instruction Level Parallelism* (<http://www.jilp.org/vol7>), April 2005.
- [6] A. Seznec. Analysis of the o-gehl branch predictor. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, June 2005.
- [7] 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.
- [8] A. Seznec and A. Fraboulet. Effective ahead pipelining of the instruction address generator. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, June 2003.
- [9] André Seznec, Stéphan Jourdan, Pascal Sainrat, and Pierre Michaud. Multiple-block ahead branch predictors. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 116–127, 1996.
- [10] André Seznec and Pierre Michaud. A case for (partially)-tagged geometric history length predictors. *Journal of Instruction Level Parallelism* (<http://www.jilp.org/vol7>), April 2006.