Five poTAGEs and a COLT for an unrealistic predictor*

Pierre Michaud
Inria

1 Summary

The predictor we submit to the competition is for exploring the limits of branch prediction, it is not intended to be implemented in hardware. The proposed predictor consists of 5 huge TAGE predictors which we have tuned for better accuracy under huge storage budget. The overall prediction is obtained by combining the 5 TAGE predictions via COLT fusion. One of the 5 TAGE components takes a conventional global path as input. The 4 other TAGE components take as input different sorts of first-level histories. On the CBP-4 traces, the proposed predictor achieves 1.782 MPKI.

2 The TAGE predictor

The predictor we submit to the unlimited-size track is based on a modified version of the TAGE predictor [12, 11]. A TAGE predictor consists of several tagged tables and one tagless table. Each entry of the tagless table contains a taken/not-taken counter. Each entry of a tagged table contains a tag, a taken/not-taken counter, and a u bit. The tagless table is indexed with the address of (conditional) branches: it provides a default prediction when none of the tagged table give a hit. The tagged tables are indexed through a hash on the branch address and the *global path* [8]. The path lenghs follow a geometric progression [10]. TAGE is based on the PPM principle [1]. In a PPM-like predictor [6], the prediction is always given by the longest hitting path. However, there are some situations where the second longest hitting path is likely to be more accurate than the longest one, especially when the longest hitting path entry is "fresh", i.e., it has just been initialized but not yet updated. TAGE exploits this fact with a single USE_ALT_ON_NA counter (acting like a meta-predictor [5]) for selecting between a fresh longest hitting prediction and the second longest hitting prediction [12]. The u bit in each entry indicates if that entry is likely to be useful or not. When the u bit is set, it protects the entry from being "stolen" [6]. All the u bits are cleared when it is detected that the predictor is "jammed" [11]. The u bit is also used to detect if an entry is fresh: an entry is considered fresh if the u bit is not set and the taken/not-taken counter value is weak [12].

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

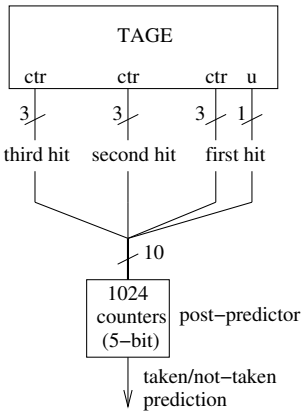

Figure 1: poTAGE: the post-predictor inputs are the u bit and taken/not-taken counter of the longest hitting TAGE entry, and the taken/not-taken counters of the second and third longest hitting TAGE entries.

3 The poTAGE predictor

3.1 Post predictor

The poTAGE predictor, shown in Figure 1, is similar to TAGE, except that we replace the USE_ALT_ON_NA mechanism with a *post-predictor* (poTAGE = post-predicted TAGE). In the submitted predictor, the taken/not-taken counter in each TAGE entry is 3-bit wide. Our post-predictor is a 1024-entry table, each table entry holding a 5-bit taken/not-taken counter. The post-predictor is indexed with the u bit and counter value of the longest hitting TAGE entry, and the counter values of the second and third longest hitting TAGE entries (10 index bits total). The 5-bit counter is used and updated like a conventional taken/not-taken counter [13].

3.2 Ramp-up

In a TAGE predictor, the update is crucial, it must be done very carefully: only the longest hitting counter is updated [6], and only a few new entries are allocated upon a misprediction for path lengths greater than the longest hitting length [12, 11]. However, because we assume a huge predictor size, it is possible to use a more aggressive update policy to decrease mispredictions due to cold-start effects:

Instead of updating only the longest hitting counter, we update all the hitting counters, whether or not the branch was correctly predicted.

Instead of allocating a few new entries, and only upon a misprediction, we allocate entries for all the path lengths greater than the longest hitting length, whether or not the branch was correctly predicted, and provided the entries can be stolen (u bit not set). We stop doing aggressive allocation for path lengths greater than 200 branches when all the hitting counters are saturated.

We use this aggressive update policy during what we call the *ramp-up* period. When the ramp-up period is over, we switch to the careful update policy implemented in the ISL-TAGE predictor [11].

The ramp-up period length is roughly proportional to the predictor size. For the submitted predictor, we set the ramp-up period to one million mispredictions[1]. A ramp-up period might also be useful in (large) real branch predictors. However the problem in a real predictor is how to detect when to start the ramp-up.

4 Beyond TAGE

So far, the most accurate branch predictors, including TAGE, belong to the family of two-level branch predictors [14, 9, 15]. Yeh and Patt provided a taxonomy of two-level branch predictors [16]. In particular, they distinguished predictors according to their first history level. The 2006 and 2011 TAGE predictors [12, 11] are "global" schemes, as their first-level history consists of a single global path shared by all the branches.

Global schemes are the most accurate two-level predictors that we know. However they have some limits. The footprint of a static branch in a particular TAGE table grows exponentially with the path length used for that table. The footprint also grows very quickly with branch entropy: the more unpredictable the branches, the larger the footprint [7]. A large footprint incurs cold-start and capacity misses in a TAGE predictor. Applications exhibiting randomness in their control flow behavior are difficult to predict by a global scheme such as TAGE not only because of their intrinsic randomness, but also because of the large footprint.

Per-address schemes, i.e., predictors that use per-branch first-level histories [16], suffer less from this problem. Indeed, a global path contains some "noise", i.e., some entropy that does not bring any correlation information but just grows the footprint [2]. In an ideal (i.e, large enough) per-address scheme, each static branch is predicted with its own *subpath*, which it does not share with other static branches. Unlike global schemes, per-address schemes cannot exploit global branch correlations, only local correlations. However, in a per-address scheme, hard-to-predict branches do not "pollute" the subpath of easy-to-predict branches.

Global and per-address schemes are two extreme points in the set of all possible two-level predictors. Yeh and Patt also introduced per-set schemes in their taxonomy[16]. Per-set schemes have been largely ignored since their introduction, until recently when Ishii et al. introduced two per-set schemes in their FTL++ predictor [3]

In a per-set scheme, there are several subpaths like in a per-address scheme, but a subpath is shared by several static branches. Subpath sharing generally exists in practical implementations of per-address schemes because of the limited storage for first-level histories. However, in a per-set scheme, subpath sharing is intentional: the goal is to exploit both local and (to some extent) global correlations, but with a smaller footprint than global schemes.

In a global scheme such as TAGE, hash functions are applied to the global path for indexing the tagged tables. Path aliasing can occur if two different global paths map to the same table entry and tag. In a global scheme, we generally try to minimize path aliasing by using "good" hashing functions. Using a per-set scheme can be seen as intentionally introducing path aliasing for decreasing the footprint. Blind path aliasing is likely to degrade the predictor's ability to exploit global correlation. Still, the hope is that the benefit of a smaller footprint outweighs this degradation. This is generally not true for all branches but for a few of them. This is why per-set schemes should not be used alone but as components of a hybrid predictor, as Ishii et al. did in the FTL++ predictor. The type of path aliasing that per-set schemes exploit is based on the assumption that branches which are statically close to each other are more likely to be correlated than distant branches. Other forms of intentional path aliasing can be imagined (cf. Section 6).

[1]The careful update policy is almost never used in our simulations.

poTAGE	spectrum type	spectrum size (# subpaths)	# tagged tables	min subpath (# branches)	max subpath (# branches)
P0	global	1	20	7	5000
P1	per address	32	19	5	2000
P2	per 128-byte set	16	19	5	500
P3	per 2-byte set	4	19	5	500
P4	frequency	8	19	5	500

Table 1: **The 5 poTAGEs. The poTAGE tables are very large. P0 is roughly twice larger than each of the other predictors. The COLT table (not listed) is also very large.**

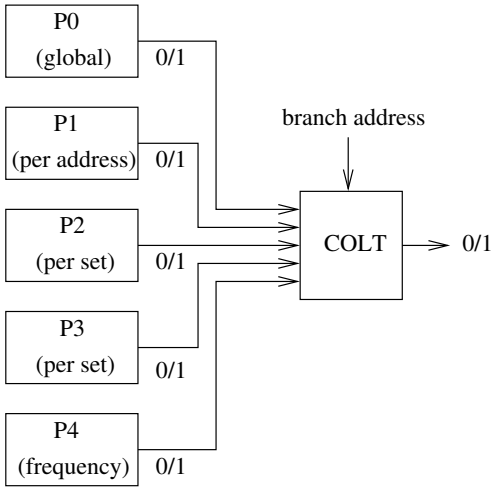

Figure 2: The multi-poTAGE predictor.

5 The multi-poTAGE predictor

The proposed predictor, called *multi-poTAGE*, is depicted in Figure 2. It is a hybrid predictor [5] combining 5 different poTAGE predictors P0, P1, P2, P3 and P4. P0 is the global path poTAGE described in Section 3. The other predictors do not use a global path:

- P1 uses 32 per-address subpaths
- P2 uses 16 per-set subpaths with 128-byte sets
- P3 uses 4 per-set subpaths with 2-byte sets
- P4 uses 8 frequency-based subpaths

Predictor P4 uses a new sort of first-level history that we describe later in Section 6.

The 5 poTAGEs are combined using COLT fusion, a method invented by Gabriel Loh and Dana Henry [4]. The COLT table is indexed with the branch address. Each COLT entry holds a 5-bit taken/not-taken counter.

Table 5 gives an overview of the 5 poTAGEs implemented in the submitted multi-poTAGE.

6 Frequency-based path spectrum

P4 uses a new sort of first-level history: frequency-based subpaths. P4 does not belong to the taxonomy of Yeh and Patt but to a more general class of two-level predictors.

In a two-level predictor, the first-level history consists of a set of subpaths that we call a *path spectrum*. At prediction time, a subpath S_p is selected from the path spectrum. Subpath S_p is used to access the second-level history (here, poTAGE), yielding a prediction. At update time, the branch updates one of the subpath S_u. In the proposed multi-poTAGE, S_p and S_u are always the same.

The spectrum of P0 consists of a single subpath shared by all the branches, i.e., it is the whole global path. The spectrum of P1 consists of 32 subpaths, and $S_p(= S_u)$ is selected with the 5 least significant branch address bits 0 to 4 (per-address first level). The spectrum of P2 consists of 16 subpaths, and $S_p(= S_u)$ is selected with branch address bits 7 to 10 (per-set first level).

The spectrum of P4 consists of 8 subpaths. The subpath $S_p(= S_u)$ is selected as follows. The branch address is used to index a Branch Frequency Table (BFT). If the BFT is large enough, each static branch uses a distinct BFT entry. Each BFT entry holds a counter indicating the current *frequency* of the static branch. The frequency of a branch is the number of times the branch has been executed until now since the counter was reset[2].

Predictor P4 seeks to exploit correlations between branches having (roughly) the same frequency. Each of the 8 subpaths $S[0]$ to $S[7]$ corresponds to a distinct frequency bin. Let F_{max} be the maximum branch frequency so far. Branches whose frequency lies in $[F_{max}/2, F_{max}]$ are predicted with subpath $S[0]$. Branches whose frequency lies in $[F_{max}/4, F_{max}/2[$ use subpath $S[1]$. Branches whose frequency lies in $[F_{max}/8, F_{max}/4[$ use subpath $S[2]$. And so forth.

Note that, after an initial period, the dynamic instances of a static branch will generally use the same subpath.

[2]In the submitted branch prediction algorithm, we reset the frequency counters only once, when the simulation starts.

7 Experimental analysis

P0 is the most accurate of the 5 poTAGEs as a single component. Compared with the USE_ALT_ON_NA mechanism used in previous TAGE predictors, the post-predictor decreases the number of mispredictions by 5% on the CBP-4 traces. Using a ramp-up period yields a further 4% decrease. To evaluate the importance of each of the 5 poTAGEs in the multi-poTAGE predictor, we selectively disabled some of them. P2 is the second most important predictor after P0, followed by P1, P4 and P3 in that order. Adding P2, P1, P4, and P3 (successively and in that order) to P0 decreases the number of mispredictions by respectively 5%, 3%, 2.5% and 1%. In total, the multi-poTAGE predictor has 10% fewer mispredictions than P0 alone.

8 Conclusion

The prediction accuracy of TAGE is very high. Still, when considering a huge storage budget for limit studies, a more aggressive update policy can be used. Global path predictors such as TAGE are theoretically able to capture all the branch correlations if the global path is long enough. However, a global path suffers from "noise" from branches that bring no correlation information but grow the footprint. To (try to) solve this issue, we combined five poTAGEs with different first-level histories, using COLT fusion. This brute force approach may not be reasonable for a realistic predictor. Future research should try to look for more cost-effective solutions.

References

[1] I.-C.K. Chen, J.T. Coffey, and T.N. Mudge. Analysis of branch prediction via data compression. In *Proc. of the 7th Int. Conference on Architectural Support for Programming Languages and Operating Systems*, 1997.

[2] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt. An analysis of correlation and predictability: what makes two-level branch predictors work. In *Proc. of the 25th Int. Symp. on Computer Architecture*, 1998.

[3] Y. Ishii, K. Kuroyanagi, T. Sawada, M. Inaba, and K. Hiraki. Revisiting local history for improving fused two-level branch prediction. In *Proc. of the 2nd JILP Workshop on Computer Architecture Competitions*, June 2011.

[4] G. H. Loh and D. S. Henry. Predicting conditional branches with fusion-based hybrid predictors. In *Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques*, 2002.

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

[6] P. Michaud. A PPM-like, tag-based predictor. *Journal of Instruction Level Parallelism*, April 2005.

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

[8] R. Nair. Dynamic path-based branch correlation. In *Proc. of the 28th Int. Symp. on Microarchitecture*, 1995.

[9] S.-T. Pan, K. So, and J. T. Rameh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proc. of the 5th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 1992.

[10] A. Seznec. Analysis of the O-GEometric History Length branch predictor. In *Proc. of the 32nd Int. Symp. on Computer Architecture*, 2005.

[11] A. Seznec. A new case for the TAGE branch predictor. In *Proc. of the 44th Int. Symp. on Microarchitecture*, December 2011.

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

[13] J. E. Smith. A study of branch prediction strategies. In *Proc. of the 8th Int. Symp. on Computer Architecture*, 1981.

[14] T.-Y. Yeh and Y. N. Patt. Two-level adaptive training branch prediction. In *Proc. of the 24th Int. Symp. on Microarchitecture*, 1991.

[15] T.-Y. Yeh and Y. N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Proc. of the 19th Int. Symp. on Computer Architecture*, 1992.

[16] T.-Y. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *Proc. of the 20th Int. Symp. on Computer Architecture*, 1993.