Looking for limits in branch prediction with the GTL predictor *

André Seznec
IRISA/INRIA/HIPEAC

Abstract

The geometric history length predictors, GEHL [7] and TAGE [8], are among the most storage effective conditional branch predictors. They rely on several predictor tables indexed through independent functions of the global branch/path history and branch address. 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.

GEHL and TAGE differ through their final prediction computation functions. GEHL uses a tree adder as prediction computation function. TAGE uses (partial) tag match. For realistic storage budgets (e.g., 64Kbits or 256Kbits), TAGE was shown to be more accurate than GEHL. However, for very large storage budgets and very large number of predictor components, GEHL is more accurate than TAGE on most benchmarks. Moreover a loop predictor can capture part of the remaining mispredictions.

We submit the GTL predictor, combining a large GEHL predictor, a TAGE predictor and a loop predictor, to the idealistic track at CBP2.

Presentation outline

In the two past years, we have introduced the geometric history length predictors, GEHL [7] and TAGE [8]. These predictors are among the most storage effective branch predictors, TAGE being more efficient than OGEHL for limited storage budgets. These predictors are natural candidates to serve as basis for the design of a limit branch predictor.

In a a preliminary study, we found that, for very large storage budget and very large number of components, the GEHL predictor, i.e. OGEHL without dynamic history length fitting [7], achieves higher accuracy than the TAGE predictor: we got respective accuracy limits of 2.842 misp/KI and 3.054 misp/KI on the distributed set of traces.

* 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).

Therefore the GTL predictor presented for the idealist track at CBP-2 is an hybrid predictor, using a very large GEHL predictor as its main component. It is combined with a TAGE predictor and a loop predictor.

In Section 1, we first briefly present the principles of the predictor components and the configurations used.

Section 2 presents the accuracy limits we found for the GTL predictor and its components.

1. The GTL predictor

The GTL predictor is illustrated on Figure 1. The GTL predictor features a GEHL predictor [7], a TAGE predictor [8] and a loop predictor as components.

The GEHL predictor is the main predictor, since it is the most accurate component. The GEHL predictor is used as the base predictor component in TAGE, i.e. when there is no partial hit on any of the TAGE components, the TAGE prediction is the GEHL prediction. Moreover, the GEHL prediction is used as part of the indices for the tagged components of the TAGE predictor as well as for the components of the metapredictor. Using a (partial) prediction as part of the index of another table was already used on the YAGS predictor [1] and the bimode predictor [4].

GTL uses a metapredictor to discriminate between the TAGE and the GEHL predictions. We use a metapredictor derived from the skewed predictor [5].

Finally, the prediction features a loop predictor. The loop predictor provides the prediction when a loop has been successively executed with 8 times the same number of iterations as in [2].

1.1. The loop predictor

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

As in [2], the loop predictor provides the global prediction when the loop has succesively been executed 8 times with the same number of iterations. The loop predictor used in the submission features 512K entries.

1.2. Common features of TAGE and GEHL

The GEHL predictor [7] and the TAGE predictor [8] rely on several predictor tables indexed through indepen-

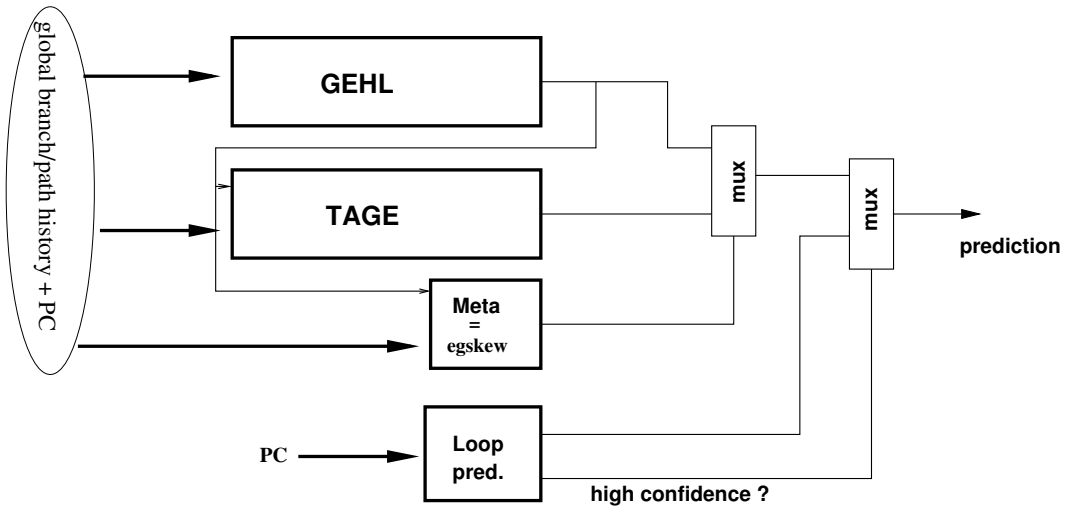

Figure 1. The GTL predictor

dent functions of the global branch/path history and branch address. On both predictors, the set of used global history lengths forms a geometric series, i.e, $L(i) = (int)(\alpha^{i-1} * L(1) + 0.5)$; table T0 is indexed with the PC. The use of a geometric series allows to use very long branch histories in the hundreds range for realistic size predictors (e.g., 64Kbits or 256Kbits). As pointed out in [7, 8], the exact shape of the series is not important. The TAGE predictor and the GEHL predictor essentially differs through their final prediction computation function.

1.3. The GEHL branch predictor [7]

1.3.1. General principle The GEometric History Length (GEHL) branch predictor features M distinct predictor tables Ti, $0 \leq i < M$ indexed with hash functions of the branch address and the global branch history. The predictor tables store predictions as signed counters. To compute a prediction, a single counter C(i) is read on each predictor table Ti. The prediction is computed as the sign of the sum S of the M counters C(i), $S = \frac{M}{2} + \sum_{0 \leq i < M} C(i)$ [1]. The prediction is taken if S is positive and not-taken if S is negative.

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 in the indexing functions for tables Ti, $1 \leq i < M$ form a geometric series.

1.3.2. Updating the GEHL predictor The GEHL predictor update policy is derived from the perceptron predictor update policy [3]. The GEHL predictor is only updated on mispredictions or when the absolute value of the computed sum S is smaller than a threshold θ. Saturated arithmetic is used. More formally, the GEHL predictor is updated as follows, *Out* being the branch outcome:

if ((p !=*Out*) or ($|S| \leq \theta$))
for each i in parallel
if *Out* then $C(i) = C(i) + 1$ else $C(i) = C(i) - 1$

1.3.3. Dynamic threshold fitting for the GEHL predictor Experiments showed that the optimal threshold θ for the GEHL predictor varies for the different applications. In [7], dynamic threshold fitting was introduced to accomodate this difficulty.

For most benchmarks there is a strong correlation between the quality of a threshold θ and the relative ratio of the number of updates on mispredictions NU_{miss} and the number of updates on correct predictions $NU_{correct}$: experimentally, in most cases , for a given benchmark, when NU_{miss} and $NU_{correct}$ are in the same range, θ is among the best possible thresholds for the benchmark.

A simple algorithm adjusts the update threshold while maintaining the ratio $\frac{NU_{miss}}{NU_{correct}}$ close to 1. This algorithm is based on a single saturated counter TC (for threshold counter).

if ((p !=*Out*) {TC= TC + 1; if (θ is saturated positive){$\theta = \theta + 1$; TC=0;} }
if ((p == *Out*) & ($|S| \leq \theta$)) {TC= TC - 1; if (θ is saturated negative){$\theta = \theta - 1$; TC=0;}}

Using a 7-bit counter for TC was found to be a good tradeoff.

1 For p-bit signed counters, predictions vary between -2^{p-1} and $2^{p-1} - 1$ and are centered on $-\frac{1}{2}$

1.3.4. GEHL configuration in the submitted predictor We leveraged the different degrees of freedom in the design of the GEHL predictor to get the best predictor that we could simulate with a memory footprint in the range of 768 Mbytes.

Experiments showed that, for a GEHL simulator with a memory footprint in the range of 768 Mbytes, using 97 tables provides a high level of accuracy [2] Experiments showed that using 8-bit counters leads to good prediction accuracy.

We also slightly improve the update policy by incrementing/decrementing twice the counters when they are between -8 and 7. This results in a gain of 0.009 misp/KI on the overall GTL predictor.

1.4. The TAGE predictor [8]

The TAGE predictor [8] is derived from Michaud's PPM-like tag-based branch predictor [6]. 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. For the submitted GTL predictor, the GEHL predictor component provides the base prediction.

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*. In this submission *u* is a 2-bit counter and *ctr* is a 5-bit counter.

1.4.1. Prediction computation on TAGE 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,on several applications, newly allocated entries provide poor prediction accuracy and that on these entries, using the alternate prediction is more efficient. 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).

Moreover, no special memorization is needed for recording the "newly allocated entry": we consider that an entry is newly allocated if 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.

2 257 would have been a marginally better choice but the simulation time was too long for CBP2 execution time constraints.

Therefore the prediction computation algorithm is as follows:

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

1.4.2. Updating the TAGE predictor We present here the various scenarios of the update policy that we implement on 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*.

Updating the prediction counters The prediction counter of the provider component is updated. For large predictors also updating the alternate prediction when the useful counter of the provider component is null results in a small accuracy benefit.

Allocating tagged entries on mispredictions For a 64 Kbits 8-component predictor [8] or 256 Kbits TAGE predictors (submissions to realistic tracks), allocating a single entry is the best tradeoff. For the very large predictor considered here, allocating all the available entries is more effective [3].

The allocation process is described below.

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

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

(B) Priority for allocation For all components Tk, $k > b$, if $u_k = 0$ then the entry is allocated.

(C) 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*).

1.4.3. TAGE configuration in the submitted predictor We leveraged the different degrees of freedom in the design of the TAGE predictor to get the best predictor that we could simulate while maintaining the total memory footprint of the simulator smaller than 1 gigabyte.

The TAGE component in the submitted predictor feature a total of 19 tagged components, the GEHL predictor is used as the base predictor. Each table feature 1M entries. The tag width is 16 bits on the tagged tables.

3 An entry is considered as free if its useful counter *u* is null.

1.5. Selecting between TAGE and GEHL predictions

As a meta-predictor to discriminate between TAGE and GEHL predictors, we found that a skewed predictor [5] works slightly better than a bimodal table (by 0.004 misp/KI). The respective history lengths of the three tables are 0, 4 and 22. As for the TAGE predictor component, the output of the GEHL predictor is used to index the meta-predictor.

For one benchmark, the GEHL+TAGE predictor exhibited a slightly higher misprediction rate than the GEHL predictor alone. Therefore to avoid this situation, we use a single safety counter that monitors that GEHL+TAGE against GEHL alone.

1.6. Information for indexing the branch predictor

1.6.1. Path and branch history The predictor components are indexed using a hash function of the program counter, the global history combining branch direction and path (7 address bits). Non-conditional branches are included in these histories.

1.6.2. Discriminating kernel and user branches Kernel and user codes appear in the traces. In practice in the traces, we were able to user code from kernal through the address range. In order to avoid history pollution by kernel code, on jumps in kernel code, we save the global branch history and path histories and restore them when jumping back in user code. Compared with using a single history, 0.074 misp/KI improvement is achieved.

1.6.3. Indexing the TAGE predictor component and the metapredictor The TAGE predictor is used to try to correct predictions of the GEHL predictor. Therefore the output of the GEHL predictor is used as part of the indices of the TAGE predictor components and the metapredictor.

1.6.4. History lengths Geometric length allows to use very long history lengths. Experiments showed that using 2,000 as the maximum history length for both TAGE and GEHL predictors is a good choice, but that using respectively 400 for GEHL and 100,000 for TAGE is marginally better (by 0.014 misp/KI)[4].

For the GEHL predictor, we force the use of distinct history lengths by enforcing the property $L(i) > L(i+1) + 1$.

1.7. Static prediction

On the first occurence of a branch, a static prediction associated with the branch opcode is used: this allows to reduce the overall misprediction rate by 0.005 misp/KI.

4 100,000 is 0.002 misp/KI better than 10,000

2. Simulation results

The average predictor accuracy of GTL is **2.717 misp/KI** on the distributed set of traces. Removing the loop predictor and the static prediction, i.e., GEHL+TAGE, one will still obtain 2.774 misp/KI, the accuracy of the GEHL predictor component alone being 2.891 misp/KI. A GEHL predictor alone using a 2,000 branch history length achieves 2.842 misp/KI.

Results for the GTL and GEHL+TAGE are displayed per application in Table 1. Results for GEHL predictor using a 2,000 branch history length are also displayed as a reference. One can note that the benefit of loop prediction is marginal apart on *164.gzip* and to a less extent on *201.compress*.

3. Conclusion

In a preliminary study, we found that, at a a very large storage budget and for very large number of components, the GEHL predictor outperforms the TAGE predictor which accuracy reaches a plateau with medium number of components (around 16) and medium storage budget (around 64 Mbytes): a 256 Kbits 13-component TAGE predictor exhibits only approximately 10% more mispredictions than the best no-storage limit TAGE predictor we found. Therefore, the GTL predictor is built around the GEHL predictor.

To grab the last pieces of predictability contained in global branch/path history, we combine GEHL and TAGE, using the GEHL prediction as the base predictor and as a partial index for TAGE.

A loop predictor can improve a little bit the prediction accuracy of the GEHL+TAGE predictor. On the set of benchmarks for CBP2, the loop predictor has only small return apart on 164.gzip.

Apart the loop predictor, and despites our best efforts, we have not found any way to integrate a local history predictor bringing any benefit to the GTL predictor.

References

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

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

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

[4] C-C. Lee, I-C.K. Chen, and T.N. Mudge. The bi-mode branch predictor. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, Dec 1997.

	164	175	176	181	186	197	201	202	205	209
GTL	9.393	7.783	2.434	7.312	1.591	3.891	5.260	0.293	0.234	2.168
GEHL+TAGE	9.992	7.785	2.472	7.350	1.602	3.901	5.397	0.302	0.247	2.175
GEHL	10.166	7.919	2.490	7.663	1.630	3.969	5.480	0.321	0.241	2.234
	213	222	227	228	252	253	254	255	256	300
GTL	0.946	0.943	0.271	0.425	0.177	0.128	1.129	0.087	0.033	9.858
GEHL+TAGE	0.991	0.990	0.287	0.442	0.180	0.137	1.211	0.093	0.041	9.890
GEHL	1.043	1.035	0.291	0.485	0.179	0.183	1.257	0.108	0.044	10.113

Table 1. Per benchmark accuracy in misp/KI

[5] 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.

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

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

[8] 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.