Exploring value prediction with the EVES predictor

André Seznec

Exploring value prediction with the EVES predictor *

André Seznec
INRIA/IRISA

Outline

In this study we explore the performance limits of value prediction for small value predictors (8KB and 32KB) in the context of a processor assuming a large instruction window (256-entry ROB), a perfect branch predictor, fetching 16 instructions per cycle, an unlimited number of functional units, but a large value misprediction penalty with a complete pipeline flush at commit on a value misprediction

This evaluation framework emphasizes two major difficulties that an effective hardware implementation value prediction will face. First the prediction of a value should be used only when the potential performance benefit on a correct prediction outweighs the potential performance loss on a misprediction. Second, value prediction has to be used in the context of an out-of-order execution processor with a large instruction window. In many cases the result of an instruction has to be predicted while one or several occurrences of the same instruction are still progressing speculatively in the pipeline. Many value predictors (FCM predictors [7], stride predictors [3]) are using the result(s) of one or more previous occurrence(s) of the instruction to predict the result of the current occurrence. In this case, a prediction cannot be used in the pipeline unless no instruction occurrence is still speculative or all the occurrences present in the pipeline have been predicted with high confidence.

Our proposition EVES, for Enhanced VTAGE Enhanced Stride, combines two predictor components which do not use on the result of the last occurrence of the instruction to compute the prediction. We use an enhanced version of the VTAGE predictor [5], E-VTAGE. On VTAGE, the predicted value is the value directly read at prediction time on the predictor tables. Second, we propose a enhanced version of the stride predictor, E-Stride. E-Stride computes the prediction from the last committed occurrence of the instruction and the number of speculative inflight occurrences of the instruction in the pipeline.

The prediction flowing out from E-Stride or E-VTAGE is used only when its confidence is high. The major contribution of this study is the algorithm to assign confidence to predictions depending on the expected benefit/loss of a prediction. For the predictor components, the predictor entry allocation and victim selection is also guided by this expected benefit/loss. The confidence/priority assignment algorithms use probabilistic counters defined by Riley et al. [6].

On the distributed traces, the EVES predictors with respectively 8KB, 32KB and unlimited storage budgets achieve respectively **4.026 IPC**, **4.202 IPC** and **4.408 IPC** (geometric mean), i.e., respectively 25.3 %, 30.8 % and 37.3 % improvement over the 3.211 IPC achieved without value prediction. Most of this benefit is brought by the 48-entry E-Stride predictor with 16.1 % speedup.

*This work was partially supported by an Intel research grant

1 The Enhanced Stride Predictor

The stride predictor [3] computes the predicted value as the result of the addition of the last result of the instruction and a stride. The stride is dynamically computed and updated at validation time. The stride is read on the predictor table while the last value is read either on the predictor table or more often when there is at least an inflight occurrence of the instruction in the pipeline from the speculative instruction window. The prediction is only used in the pipeline when both the stride and the last value are high confidence. That is if any of the inflight occurrences present in the pipeline was not high confidence then one can not use the prediction: when the stride becomes high confident for instruction I, one cannot begin using the hardware value predictions when before all the occurrences of instruction I have disappeared from the instruction window. On short loops, this might never happens unless the pipeline is flushed. *Notice that even on a realistic framework implementing branch prediction or memory order violation disambiguation, complete flushes of the pipeline might not be that frequent since these incidents are handled after execution stage, not commit stage.*

Dealing with speculative occurrences In order to avoid the above mentioned issue, we introduce a modified version of the stride prediction algorithm. The predicted value for instruction I is computed as the Last **Committed** Value added with $(Inflight + 1)$*Stride where $Inflight$ is the

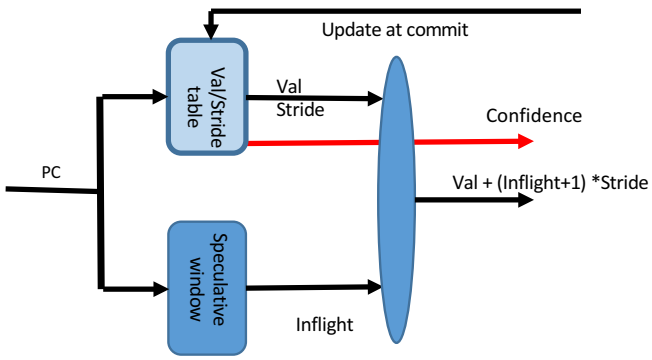

Figure 1. Predicting with E-Stride predictor

number of speculative occurrences of instruction I. As soon as the stride associated with instruction I reaches high confidence, the prediction can be used even if other (not predicted) occurrences of the same instruction are still present in the pipeline. This is illustrated in Figure 1. The same algorithm was suggested to use value prediction for predicting branches [1].

A limited storage budget E-stride predictor In the presence of VTAGE, there is no interest in using the stride predictor to predict sequences of constant values; therefore entries with null strides are systematic targets for replacement in E-Stride. A 3-way skewed-associative 48-entry E-Stride predictor was found to reach performance very close to the one of an unlimited size E-stride predictor on the distributed traces. Moreover the width of the stride can be reduced to 20 bits.

Confidence management and entry allocation Not all predictions are equal. Predicting the result of a load missing the LLC cache is more likely to lead to a performance benefit than predicting the result of a single cycle ALU operation.

To manage confidence 5-bit probabilistic counters [6] are used. A prediction is forwarded to the pipeline only if the confidence counter reaches 7.

We carefully define the probabilities of incrementing the confidence counter depending on the instruction type, the instruction behavior and the stride value. In our submission, the probability of incrementing the counter on a load is 1 on a miss on the LLC, $\frac{3}{4}$ on a hit on the LLC, $\frac{5}{16}$ on a hit on the L2 cache, $\frac{9}{64}$ on a hit on the L1 cache. For a slow ALU or a floating point operation, probability $\frac{1}{32}$ is used. For single cycle ALU instructions, probability $\frac{1}{128}$ is used. Finally we remarked that small strides (e.g. 1) does not bring the same benefit as large ones; therefore for small strides, the confidence counter is incremented with a lower probability.

The probability of inserting an entry on a miss on the predictor is also adjusted to the potential performance benefit/loss expected from predicting the instruction. This probability varies from 1 for a load missing on the LLC to $\frac{1}{16}$ for a load hitting in the store queue and to $\frac{1}{64}$ for a single cycle ALU instruction.

Abruptly resetting the confidence counter on a misprediction often leads to successions of warming periods with no benefits from stride predictions followed by a period of useful predictions. Therefore we opted for a decrease of the confidence counter by 4. With a 5-bit counter and a high confidence threshold of 7, in most cases the use of the prediction can restart immediately after a misprediction.

On an entry allocation, the selection of the target is also guided by the potential benefit/target from predicting this target entry. Each entry is protected by a 2-bit *useful* counter. This probabilistic counter is incremented on a correct prediction as the confidence counter, reset on a misprediction, and smoothly aged on tries to steal the entry.

Avoiding some slowdowns Several traces were encountering slowdowns with the E-Stride predictor. In order to avoid/decrease this phenomenon, we implemented a simple safety net. The ratio of misprediction associated with E-stride on the ratio of instructions is monitored and maintained under $\frac{1}{1024}$ through a 16-bit counter (SafeStride in the code).

2 Enhanced VTAGE

VTAGE was introduced in [5] and is directly derived from the indirect branch predictor ITTAGE [8].

VTAGE uses the PC and the branch history to predict the instruction result. VTAGE uses several tables for storing predictions. Each table is indexed by a different number of bits of the global branch history, hashed with the PC of the instruction. The different lengths form a geometric series [8]. The tables are backed up by a base predictor – a tagless Last Value Predictor [2]? which is accessed using the instruction address only. In VTAGE, an entry of a tagged component consists of a partial tag, a usefulness counter *u* used by the replacement policy, a full 64-bit value *val*, and a confidence counter *c*. At prediction time, all components are searched in parallel to check for a tag match. The matching component accessed with the longest history provides the prediction to the pipeline.

We describe below the VTAGE features that we modify in E-VTAGE.

Enhancements over VTAGE

Tags and associativity on the PC indexed component A difference between value prediction on VTAGE and indirect branch prediction on ITTAGE is usage. An indirect branch predictor must deliver a prediction otherwise the processor stops. The value predictor has to deliver a prediction on every access. In practice, even when it delivers a value, the predicted value is used in the pipeline only if it has high confidence.

The first modification we make over the initial VTAGE proposition is related to this property. On E-VTAGE, we implement tags on all the components; moreover we add associativity to this component (2-way skewed associative).

Reducing the data volume Since only high confidence predictions are used, many values stored in the predictor are never used as predictions in the pipeline. As a consequence, storing 64 bits of data value in each VTAGE entry is a waste of storage space. The second modification we make over the initial VTAGE is to initially store only a hash of the value in the E-VTAGE entry. When the confidence of this entry reaches a level close to high confidence, we replace this hash by a pointer to a table where we store effective 64-bit values, i.e. E-VTAGE predicts a pointer in the value table (Figure 2).

For instance, on the 32KB predictor, each entry features a 11 bit hash-or-pointer field instead of a 64-bit value field and we use a 3-way skewed associative 1536-entry 55-bit value array. In case of a high confidence prediction, the 64-bit value is reconstructed with the index (9-bits) and the 55-bit value. This allows us to implement a more than 6K-entry E-VTAGE in a 32KB budget instead of a 3K-entry VTAGE if each entry had featured a complete 64-bit value.

Sharing the E-VTAGE storage space We assume a banked interleaved E-VTAGE. The overall storage is shared among all the logical tables of E-VTAGE. The weird numbers of banks in the submitted predictors are due to the fixed storage budget constraints imposed by the championship rules.

Careful management of confidence As already mentioned for E-Stride, one should only use the predicted value when a performance benefit can be expected in average. Such a benefit depends on the potential performance benefit on a correct prediction, on the potential performance loss on a misprediction and the probability of misprediction. Clearly the tradeoff is dependent on the specific instruction.

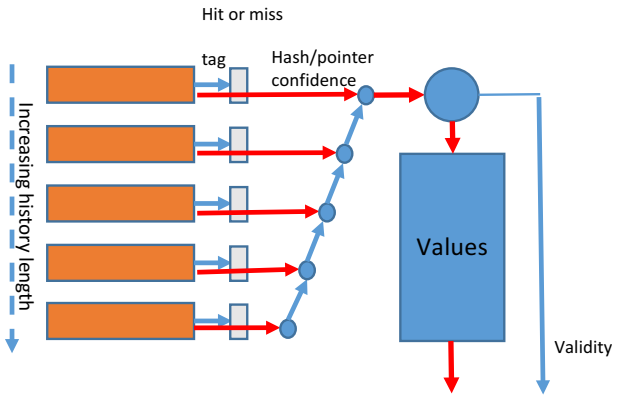

Figure 2. The E-VTAGE predictor

For the submitted E-VTAGE, we use the probabilistic 3-bit confidence counters as in the initial VTAGE proposition [5].

We carefully defined the updating policy of confidence counters on correct predictions and mispredictions depending on the instruction type, on the provider component, but also on the actual value. In practice, the average benefit of predicting a null value (or a small value) is lower than the average benefit of predicting a large value.

If the provider component is the index based component then the potential benefit is higher than if it is the provider is a tagged component. Therefore the probability of incrementing the confidence counter varies in a large range from 1 for a large value prediction hitting on the index based table and missing the LLC, to $\frac{1}{256}$ for a simple ALU operation with null result and a hit on an history indexed component. This set of probabilities was also slightly adapted to the predictor sizes (see the predictor codes).

The algorithm for updating the confidence counter on a misprediction must also be carefully optimized. If the prediction is high confidence then the counter is not directly reset, but just decremented by 2 otherwise it is reset. This provides some hysteresis, thus allowing the use of the prediction quite rapidly after an "accidental" misprediction.

Allocations on misses and mispredictions TAGE and ITTAGE [8] rely on (quasi)-systematic allocation of new entries after a misprediction. On a limited storage budget E-VTAGE predictor, such a strategy would lead to a huge majority of allocations of entries that do not reach high confidence before being evicted.

Therefore on misses, we only allocate a new entry on E-VTAGE with a probability depending on the instruction type, its latency and its actual result. For instance, on the 32 KB predictor, this probability varies from $\frac{1}{2}$ for a large

range value load missing on the LLC to $\frac{1}{2048}$ for a single cycle ALU on a null value.

The choice of a victim for replacement is based on a 2-bit useful field *u* as on TAGE. *u* is reset on mispredictions, but also through a resetting algorithm inspired from the one implemented for TAGE. The *u* incrementing algorithm is also designed to favor the entries that are likely to result in high benefit predictions (probabilities $\frac{1}{32}$ to 1, see code).

Dealing with bursts of mispredictions To avoid burst of mispredictions, E-VTAGE predictions are not used if an E-VTAGE misprediction has occurred in the last 128 instructions. This does not really increase performance, but reduces the number of mispredictions.

3 Predictors budgets and characteristics of the submitted predictors

3.1 The 8KB EVES predictor

The E-VTAGE predictor necessitates **60274** bits:

- a 3-way skewed associative data value table with a total of 384 entries. Each entry is composed of a 57-bit value and a 2-bit useful counter: **22656 bits**
- a 47 -bank interleaved VTAGE array with 32 entries on each bank. Each entry consists of a 9 bit hash-or-pointer, a 11-bit tag, a 2-bit useful counter and a 3-bit confidence counter, i.e., 25 bits per entry : **37600 bits**
- A **10-bit** counter TICK to manage the resetting of the useful counters and a **8-bit** counter to date the last misprediction.

A 30-bit global path history is used for E-VTAGE, i.e. the addresses of 30 taken branches are taken into account.

The E-stride predictor is 3-way skewed associative. It features a total of 48 entries. Each entry is composed of a 64-bit last value, a 2-bit useful counter, a stride limited to 20 bits, a 14-bit tag, a 5-bit confidence counter, and a 1 bit *NotFirstOcc* indicating that the stride has been set for the entry since its allocation or the last misprediction. Each entry features a total of 106 bits, i.e. a total of **5088 bits** for the 48 entries and 16 bits for the SafeStride counter, i.e., a total of **5104 bits**.

The 8KB EVES predictor necessitates a total storage budget of **65378 bits**.

3.2 The 32 KB EVES predictor

The E-VTAGE predictor necessitates **256914** bits:

- A 3-way associative data value table with a total of 1536 entries. Each entry is composed of a 55-bit value and a 2-bit useful counter: **87552 bits**.
- A 49-bank interleaved VTAGE array with 128 entries on each bank. Each entry consists of a 11-bit hash-or-pointer, a 11-bit tag, a 2-bit useful counter and a 3-bit confidence counter, i.e., 27 bits per entry : **169344 bits**.
- A **10-bit** counter TICK to manage the resetting of the useful counters and a 8-bit counter to date the last misprediction.

A 127-bit global path history is used for E-VTAGE.

The E-stride predictor is the same as the one for the 8KB configuration, i.e., **5104 bits**.

The 64KB EVES predictor necessitates a total storage budget of **262108 bits**.

3.3 The unlimited size EVES predictor

We use quasi-illimited size predictor components with millions of entries and a 511-bit global path history for E-VTAGE.

4 Performance analysis of the EVES predictor

On the distributed traces, the EVES predictors with 8KB, 32KB and unlimited storage budgets achieve respectively **4.026 IPC**, **4.202 IPC** and **4.408 IPC** (geometric mean), i.e., respectively 25.3%, 30.8% and 37.3 % improvement over the 3.211 IPC achieved without value prediction.

E-Stride versus E-VTAGE We also simulated independently E-VTAGE and E-Stride.

The 48-entry E-Stride predictor achieves 16.1 % performance improvement by itself. This is particularly impressive for such a small structure (less than a 1 Kbytes !!).

The E-VTAGE components from the 8KB and 32KB EVES achieve 7.0% and 11.2 % performance improvement respectively.

In practice, E-VTAGE and E-Stride do not predict the same instructions. Interestingly, E-VTAGE is able to predict some instructions with stride behavior and short sequences that E-Stride does not capture. An early (and more complex version) of E-Stride was capturing these instructions, but was necessitating regular behavior (i.e., loops with constant number of iterations). In practice, the benefits of E-VTAGE and E-Stride are slightly more than multiplicative (1.112*1.161= 1.291, slightly less than 1.308).

E-Stride and E-VTAGE do not perform well on the same set of traces. For instance, on the 32 KB predictor, among the 135 traces, E-Stride achieves more than 10 % IPC improvement on 41 traces, E-VTAGE achieves the same on 40 traces, but only 19 traces belong to both sets. EVES achieves more than 10 % improvement on 71 traces, more than 20 % on 54 traces and more than 100 % on 15 traces with a peak at 1090 % on compute_int_45.

In practice, the E-Stride component is sometimes able to predict the results of loads that miss the L2 cache and sometimes the LLC cache. The coverage of the 48-entry E-Stride component is extremely dependent on the traces varying from 0 to up to 56.4 % with an arithmetic mean of only 3.4 %.

The coverage of the E-VTAGE component is much higher (18.7 % in average for the 32 KB predictor, 45.3 % for the unlimited size predictor) with several traces experiencing coverage higher than 60 % (up to 77 %). However this higher coverage does not translate in the same performance benefit per used prediction as on the E-Stride predictor, since the E-VTAGE predicts much less high reward values (i.e, LLC misses or L2 misses) than the E-Stride predictor.

Using the number of inflight occurrences on the E-Stride predictor Relying on the number of inflight occurrences to compute the prediction rather than on the last (potentially) predicted value allows to increase the number of predictions used in the pipeline on the stride predictor. Our simulation showed that this improves the performance by about 2 %.

Impact of confidence updating policy and entry allocation policy To illustrate the benefit of using differentiated probabilities, we run the exact same configuration as EVES, but with fixed confidence update probability and fixed entry allocation probability ($\frac{1}{32}$). This resulted in a performance of 3.710 IPC instead of 4.202 IPC on the 32 KB predictor.

Should we predict only loads? We run simulations with predicting only loads. This reduces the performance improvement by 2.7 % for the unlimited configuration, by 1.3% for the 32 KB configuration and 1.5 % for the 8KB configuration. This confirms that most of benefit from value prediction can be obtained from predicting only load results as pointed out in [9].

5 A few remarks on the ChampionShip framework

The reader should not assume that since the simulation framework is an idealistic framework, the potential benefits of value prediction are systematically overestimated in this study.

We would like to point out two features that lead to underestimate the potential benefits of value prediction by the Championship framework.

First the framework assumes that after an instruction triggering a value misprediction, no instruction is fetched and speculatively executed. The instructions that follow the mispredicted instruction are delayed till the commit of the mispredicted instruction. On an effective hardware implementation, some of these instructions would have already been executed at the commit of the mispredicted instruction; particularly load instructions missing the memory hierarchy could have already triggered the fetch of the data from the main memory or the LLC. Thus the simulation framework overestimates the penalty on some mispredictions.

Second, the framework is not providing the precise OpCode of the instructions. With the EOLE architecture [4], it has been shown that a large portion of instruction results could be computed in the front-end from computed or predicted registers; thus increasing considerably the number of registers that can be already computed/predicted before entering the execution core. For instance, assuming the 32 KB EVES predictor, 40 % of the register results could have been computed/predicted against 22 % predicted with the EVES predictor.

6 Summary

EVES combines two predictors E-VTAGE and E-Stride addressing two different instruction behaviors. The E-Stride predictor has no need for the speculative result of the last occurrence of the instruction, but can rely on the last committed value *and* the number of inflight instances of the instruction in the speculative window. E-VTAGE is a storage effective VTAGE predictor which stores full 64-bit values only when they become very likely to be used in the pipeline.

For these two predictor components, we have developed an efficient approach for managing the prediction uses in the pipeline and and for allocation/eviction of predictor entries. We differentiate the probabilities of updating confidence counters and useful counters and the probabilities of predictor insertions depending on instruction type, instruction latency and instruction result value. On an effective hardware implementation, these probabilities will have to be carefully designed, depending on the management of mispredictions (at execution or at commit), the effective characteristics of the pipeline (execution width, ...), the branch predictor behavior, the criticality of the instruction, the memory level parallelism, ...

Despites the limited information (instruction type, actual latency, actual value) that was available from the simulation framework, the submitted EVES predictors demonstrate a large potential performance impact of value prediction with relatively simple prediction algorithms and limited storage budget.

This study has been focused on limited size predictors. Therefore we have only explored the limits of value prediction with very large storage budgets with EVES. Other prediction schemes could be considered to supplement E-Stride and E-VTAGE such as predicting load addresses [9] or Finite Context Methods [7].

References

[1] Timothy H. Heil, Zak Smith, and James E. Smith. Improving branch predictors by correlating on data values. In *MICRO*, 1999.

[2] M.H. Lipasti and J.P. Shen. Exceeding the dataflow limit via value prediction. In *Proceedings of the Annual International Symposium on Microarchitecture*, pages 226–237. IEEE Computer Society, 1996.

[3] A. Mendelson and F. Gabbay. Speculative execution based on value prediction. Technical Report TR1080, Technion-Israel Institute of Technology, 1997.

[4] Arthur Perais and André Seznec. EOLE: Paving the Way for an Effective Implementation of Value Prediction. In *International Symposium on Computer Architecture*, volume 42, pages 481 – 492, Minneapolis, MN, United States, June 2014. ACM/IEEE.

[5] Arthur Perais and André Seznec. Practical data value speculation for future high-end processors. In *International Symposium on High Performance Computer Architecture*, pages 428 – 439, Orlando, FL, United States, February 2014. IEEE.

[6] N. Riley and C. B. Zilles. Probabilistic counter updates for predictor hysteresis and stratification. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 110–120, 2006.

[7] Y. Sazeides and J.E. Smith. The predictability of data values. In *Proceedings of the Annual International Symposium on Microarchitecture*, pages 248–258. IEEE, 1997.

[8] A. Seznec and P. Michaud. A case for (partially) tagged geometric history length branch prediction. *Journal of Instruction Level Parallelism*, 8:1–23, 2006.

[9] Rami Sheikh, Harold W. Cain, and Raguram Damodaran. Load value prediction via path-based address prediction: avoiding mispredictions due to conflicting stores. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2017, Cambridge, MA, USA, October 14-18, 2017*, pages 423–435, 2017.