Probabilistic Counter Updates for Predictor Hysteresis and Bias

Nicholas Riley and Craig Zilles
Department of Computer Science
University of Illinois at Urbana-Champaign
Email: {njriley, zilles}@uiuc.edu

***Abstract*— Hardware predictor designers have incorporated hysteresis and/or bias to achieve desired behavior by increasing the number of bits per counter. Some resulting proposed predictor designs are currently impractical because their counter tables are too large. We describe a method for dramatically reducing the amount of storage required for a predictor's counter table with minimal impact on prediction accuracy. Probabilistic updates to counter state are implemented using a hardware pseudo-random number generator to increment or decrement counters a fraction of the time, meaning fewer counter bits are required.**

We demonstrate the effectiveness of probabilistic updates in the context of Fields *et al.*'s critical path predictor, which employs a biased 6-bit counter. Averaged across the SPEC CINT2000 benchmarks, our 2-bit and 3-bit probabilistic counters closely approximate a 6-bit deterministic one (achieving speedups of 7.75% and 7.91% compared to 7.94%) when used for criticality-based scheduling in a clustered machine. Performance degrades gracefully, enabling even a 1-bit probabilisitic counter to outperform the best 3-bit deterministic counter we found.

I. Introduction

Microarchitectures of modern processors employ speculation to improve performance by exploiting predictability. The output of hardware predictors determines which, if any, speculative actions to perform. Predictors may consist of one or more counters, whose states are updated as relevant behavior is observed.

In predictors that record the behavior of individual instructions, an array of counters maintains independent information for each of a set of instructions. Typically, the array is indexed by a hash of the instruction PC. When an instruction exhibits one class of relevant behavior, the corresponding counter is incremented; when it does something else, the counter is decremented. Comparison of the counter value with a threshold value yields a binary prediction.

While 2-bit counters are often sufficient to predict branch outcomes [11], this paper focuses on predictors used for speculation control, which often include larger counters to provide additional hysteresis and to enable biased updates.

Speculation control predictors, such as a proposed selective value predictor [1], have designs reflecting asymmetric misprediction penalties: the performance lost by speculating with an incorrect value (often incurring a pipeline flush) is significantly greater than the opportunity cost of not speculating if the proposed prediction was correct. This imbalance motivates a *biased* counter design, which conservatively prefers a cheaper false negative to a more expensive false positive.

Manuscript submitted: 30 June 2005. Manuscript accepted: 19 August 2005. Final manuscript received: 23 August 2005.

Two attributes typify existing biased counter designs. First, the magnitudes of counter increment (I) and decrement (D) differ (for example, $I = 1$ and $D = 2$). In this way, the counter can be configured to saturate when the correct-to-incorrect prediction ratio is greater than 50%: the required ratio is $\frac{D}{I+D}$. Second, additional bits per counter store more history. The resulting *hysteresis* serves as a low-pass filter, thereby ignoring occasional chance instances of correct predictions.

We represent a generalized counter as a 4-tuple (n, I, D, T): an n-bit saturating counter's value is incremented by I on a positive outcome, decremented by D on a negative outcome, and predicts true if greater than the threshold value T, where possible counter values range from 0 to $2^n - 1$. The JRS resetting counter for branch confidence [4] is thereby represented as $\langle 4, 1, 15, T\rangle$, a confidence counter for selective value prediction [1] as $\langle 4, 1, 7, \{2, 6, 14\}\rangle$ (low, medium and high confidence thresholds T, respectively), and Fields's criticality predictor [2] as $\langle 6, 8, 1, 8\rangle$.

Large-counter tables can require significant amounts of fast memory. For example, Fields's proposed critical path prediction table consists of 16K 6-bit counters, a 12 KB array. Power, area and latency concerns may make auxiliary predictors of this size difficult to justify, even in future microarchitectures.

In this paper, we demonstrate how to implement counter bias and hysteresis using fewer bits of storage, achieving equivalent critical path prediction accuracy with a predictor one-third the size. Our key insight is that bias and hysteresis can be achieved without large counters by replacing some of the bits in each counter by probabilistic updates.

Probabilistic counter updates correspond to fractional values for I and/or D. For example, $D = \frac{1}{4}$ means that the counter has a 25% chance of being decremented on negative feedback. A pseudo-random number generator (PRNG), whose output is compared with fixed threshold values, can control the frequency of counter increments and/or decrements. We can approximate a 6-bit predictor $\langle 6, 8, 1, 8\rangle$ with a 3-bit predictor $\langle 3, 1, \frac{1}{8}, 1\rangle$, or even a 2-bit predictor $\langle 2, \frac{1}{2}, \frac{1}{16}, 0\rangle$.

In the following section, we identify the components of our strategy: the linear feedback shift register—a simple PRNG—and probabilistic counter updates. Section III describes the

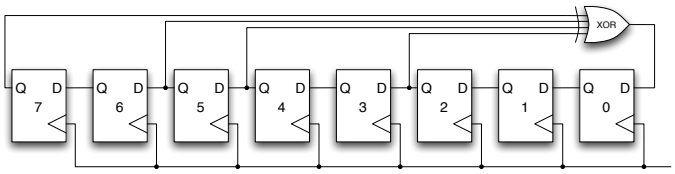

Fig. 1. An 8-bit LFSR with taps at bits 2, 4, 5 and 7. The least significant, or most recently generated, bit is numbered 0. A LFSR with maximum period always has an even number of taps; the oldest bit is always tapped.

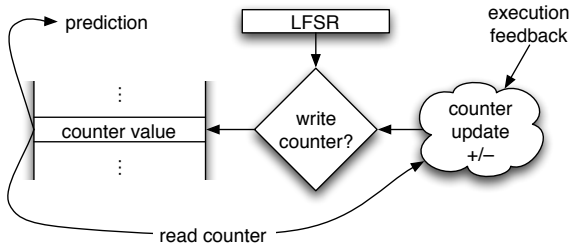

Fig. 2. Probabilistic, LFSR-mediated counter updates.

context of critical path prediction and presents our experiments, which compare the accuracy and performance of traditional deterministic critical path predictors with probabilistic predictors. While our evaluation focuses on criticality prediction, we briefly discuss experiments on branch confidence predictors and the applicabilty of our techniques to branch outcome prediction.

II. Design

A. Linear Feedback Shift Register

A linear feedback shift register [3] is a sequential shift register with logic causing it to cycle through a pseudo-random sequence of values. When optimally configured, an n-bit LFSR has a period of $2^n - 1$.

The feedback in a LFSR is generated by the output of a set of its stages, called *taps*, which are XORed into a value shifted into the register. The initial value in the register may be set from 1 to $2^n - 1$ to start the sequence at a given position. A shift occurs on each clock cycle. Fig. 1 depicts an 8-bit LFSR with taps in an optimal configuration.

Our application does not require high-quality random numbers. While a LFSR's behavior is extremely predictable, it is adequate for probabilistic counter updates. Use of a 32-bit Mersenne Twister-based PRNG [7] yielded prediction results that did not differ substantially from those obtained with a 16-bit LFSR. Given this result, we did not explore more sophisticated PRNG hardware.

B. Probabilistic Counter Updates

Multi-bit counters are updated in a read-modify-write sequence. For traditional, deterministic counters, the current value of the counter is read, incremented by I or decremented by D (using saturating arithmetic) depending on the type of event, and written back to the counter.

Fig. 2 illustrates our proposed probabilistic counter update mechanism. For fractional values of I or D, the counter write is conditional. If the value read from the LFSR exceeds a threshold value—if I and D differ and are both fractional, two threshold values are required—the counter is updated.

III. Results

We chose Fields *et al.*'s critical path predictor [2] to demonstrate the effectiveness of probabilistic counter updating. In this model, a token-passing predictor updates biased 6-bit counters. The output of the predictor is used to drive optimizations; the results of the optimizations may affect future predictions. In practice, predicting a few non-critical instructions as critical does not greatly affect performance, but missed critical instructions represent a significant loss of optimization opportunity. As a result, the counters are biased to predict an instruction as critical if it has recently been measured as critical at least $\frac{1}{9}$ of the time.

The counters are sized to provide significant hysteresis, because the application of critical path optimizations often moves instructions off the critical path, though the instructions should continue to be predicted as critical. We experimented with reducing the size and/or bias of these deterministically updated counters; the corresponding reductions in prediction accuracy that resulted justify Fields's selection of counter parameters. For comparison, we include the best deterministic 3-bit predictor we found, $\langle 3, 4, 1, 4 \rangle$.

The following experiments evaluate 3, 2 and 1-bit probabilistic counters which represent linear scalings of Fields's deterministic counter.[1] When reducing a 6-bit counter to a 3-bit probabilistic one, the values for I, D and T are scaled down by a factor of 2^3. Fractional values of T were rounded down to 0, as this provided better performance. The 1-bit probabilistic counter is not suggested as a practical alternative: its behavior exhibits the graceful degradation of our method.

We describe two sets of experiments performed using critical path predictors. The first demonstrates that our proposed probabilistically updating counter designs have the same qualitative behavior as deterministically updated counters two to three times larger. The second demonstrates that this behavior translates into equivalent performance for criticality-based scheduling in a clustered microarchitecture.

Experimental Method: Our simulated 8-wide dynamically-scheduled superscalar processor has a 128-entry instruction window, 13-stage pipeline, and a *gshare* branch predictor using 16 bits of global history. We assume a perfect instruction cache, a 32 KB, 4-way L1 data cache (2-cycle access), a 1 MB, 8-way L2 data cache (10-cycle access), and memory with a 100-cycle latency and 4-cycle interconnect occupancy for 32-byte blocks. Simulated 16-bit LFSRs are initialized at the start of each experiment; one shift occurs per value read from the register, not per CPU cycle.

[1] As part of this work, we explored a variety of counter parameters. Interestingly, we found a 2-bit configuration $\langle 2, 1, \frac{1}{16}, 0 \rangle$ whose average performance slightly exceeds that of the baseline 6-bit predictor.

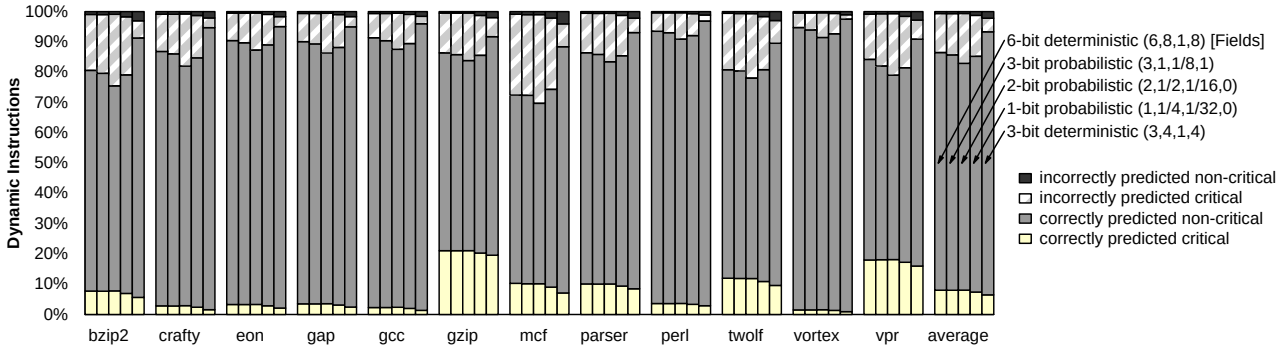

Fig. 3. Predictions from Fields's deterministic counter, three LFSR-based probabilistic counters, and a 3-bit deterministic counter trained on model-based criticality results, one static instruction at a time, using traces of 19 million dynamic instruction executions for each benchmark.

The SPEC CINT2000 benchmarks were compiled using the Digital Alpha compiler with full traditional optimization (no profile feedback). Speedups were calculated from the average of three 10 million instruction runs of the benchmarks after skipping 3, 5, and 8 billion instructions, after warming up predictors and caches for one million instructions.

A. Predictor Behavior

To evaluate the behavior of alternative counter designs in isolation, we examined simulation-generated traces of Fields's model-based criticality results grouped by static instruction. We demonstrate the prediction accuracy of probabilistic counters in the context of an alias-free predictor with immediate updates—a best-case scenario for the predictors.

The accuracy of each predictor is plotted in Fig. 3. As accurate identification of critical instructions is the greater performance contributor, the reader should focus on the bottom and top segments of each column, labeled "correctly predicted critical" and "incorrectly predicted non-critical". There is no visibly discernible difference among the 6-bit deterministic, 3-bit and 2-bit probablistic predictor results for these segments. The 1-bit predictor is noticeably less accurate, though its behavior is surprisingly similar.

All the predictors can trivially classify instructions which exhibit consistent behavior over the duration of the program: those which are always critical or never critical. More interesting cases, where instructions are occasionally critical or go through phases of criticality, distinguish the predictors' performance. The example in Fig. 4 illustrates how probabilistic counters closely approximate Fields's deterministic counter design in such a case.

The scales of the counters have been normalized to that of the baseline counter, from 0 to 63 (2^6-1). For the probabilistic counters, plateaus indicate periods when criticality feedback is being ignored. As the 3-bit counter is deterministically incremented, the corresponding upward movement in the figure corresponds directly to the 6-bit counter. The probabilistic counter has proportionally fewer intermediate steps than the 6-bit counter and so must periodically—one-eighth of the time—make correspondingly larger jumps downward.

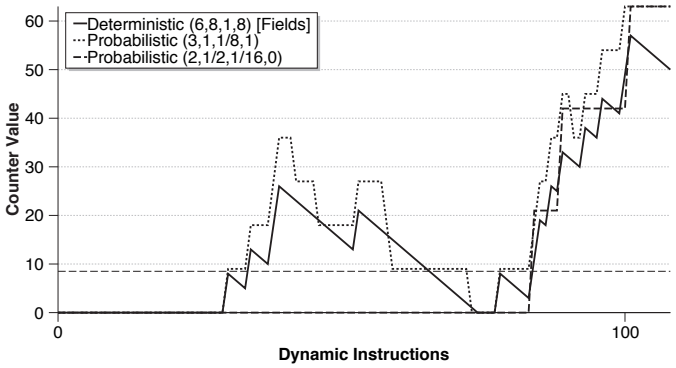

Fig. 4. Comparing values of Fields's deterministic and two approximating probabilistic counters when predicting model-based criticality of a single static instruction over a partial execution of *crafty*. The horizontal dashed line indicates the threshold of Fields's counter; the probabilistic counter values are scaled such that their thresholds are equivalent.

To demonstrate a case in which the probabilistic and deterministic counter behavior differs, we have selected a trace where the 2-bit counter—which, unlike the 3-bit counter, has a fractional value for *I*—has bad luck. In effect, it receives "tails" on flips of five coins in a row and, as a result, completely misses the initial critical behavior of the instruction. These occurances are rare, as demonstrated by Fig. 3, but lead to a slight reduction in accuracy relative to the 6-bit predictor.

B. Performance

Fields *et al.* explore two applications of their criticality predictor. To measure the performance impact of using a reduced-size probabilistic counter, we replicate their experiments with criticality-based scheduling in clustered architectures. Our baseline machine includes eight 1-wide clusters, each with 16-entry instruction windows, and uses a dependence-based steering policy. The criticality predictors employ 16K-entry tables trained using Fields's criticality model.

Fig. 5 plots speedups over executions without critical scheduling. Consistent with the idealized predictor accuracy, performance degrades slowly as the predictor storage is reduced. On average, the 2- and 3-bit counters achieve speedups only slightly smaller than the 6-bit counter: 7.75% and 7.91%

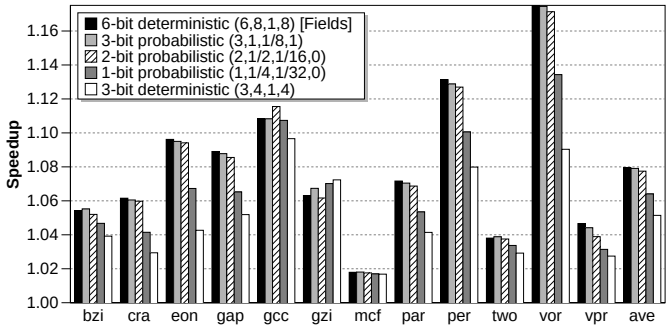

Fig. 5. Speedups over a clustered microarchitecture without critical scheduling, using predictions from model-based criticality results.

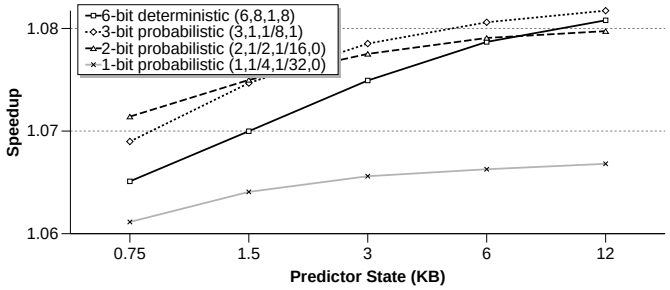

Fig. 6. Speedups for predictors of varying sizes. The 3-bit deterministic predictor (not shown) always offers less than 6% speedup.

compared to 7.94%. For some benchmarks, the probabilistic counter "gets lucky" and outperforms the 6-bit deterministic predictor: see *bzip2*, *gzip* and *twolf*. Even the 1-bit counter is competitive in several benchmarks, achieving 6.4% speedup on average. Notably, the 1-bit probabilistic predictor outperformed the deterministic 3-bit predictor, which achieved an average speedup of 5.1%.

Alternatively, keeping the predictor size constant, nondeterminism can be used to increase performance. Fig. 6 shows that for any predictor size, a probabilistic predictor exists which outperforms Fields's 6-bit deterministic one. The best probabilistic counter varies with size: the 2-bit predictor performs better at smaller sizes where conflicts from aliasing are more significant than the loss in resolution.

C. Other Applications

To demonstrate the generality of our method, we compared the performance of the JRS 4-bit resetting branch confidence estimator with LFSR probabilistic variants. Fig. 7 is patterned after Jacobsen's Fig. 8 [4]; the 3-bit probabilistic counter slightly outperforms the deterministic version, and the 2-bit counter does almost as well.

Probabilistic updates appear to require one bit of hysteresis is provided by counter state—they are ineffective at best when applied to 2-bit counters for branch prediction.

IV. Related Work

The proposed design for the Alpha EV8 branch predictor [10] employed separated prediction and hysteresis arrays.

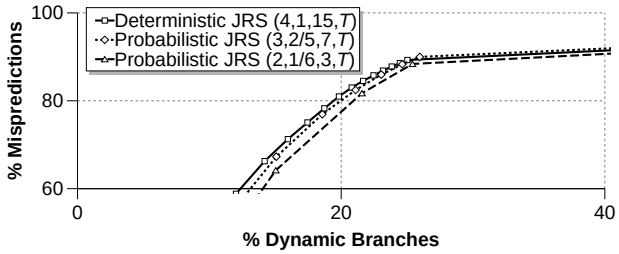

Fig. 7. Probabilistic branch confidence counters have the same behavior as deterministic counters but require less storage.

In two of the hysteresis arrays, hysteresis bits are shared between two counters. A partial update strategy, where the hysteresis table is not written on every prediction, somewhat alleviates the aliasing effects caused by shared hysteresis. Loh et al. also explore sharing hysteresis bits between counters in branch predictors [5].

Probabilistic counter updates have been proposed to reduce the size of SRAM network packet counter caches from potentially 64 to 11 bits per counter [9]. The Bandwidth Adaptive Snooping Hybrid (BASH) cache coherence protocol [6] uses a saturating policy counter $\langle 8, 1, 1, T \rangle$, where T is a LFSR-generated 8-bit integer, to determine whether to unicast or broadcast based on the bandwidth utilization. RegionScout [8] is an cache coherence optimization which uses an array of LFSRs to record locally cached regions, though the exact mechanism is not described.

References

[1] B. Calder, G. Reinman, and D. M. Tullsen, "Selective value prediction," in *Proc. 26th International Symposium on Computer Architecture*, Atlanta, GA, 1999, pp. 64–74.

[2] B. A. Fields, S. Rubin, and R. Bodik, "Focusing processor policies via critical-path prediction," in *Proc. 28th International Symposium on Computer Architecture*, Göteborg, Sweden, 2001, pp. 74–85.

[3] S. W. Golomb, *Shift Register Sequences*, 2nd ed. Laguna Hills, CA: Aegean Park Press, 1982.

[4] E. Jacobsen, E. Rotenberg, and J. E. Smith, "Assigning confidence to conditional branch predictions," in *Proc. 29th Annual International Symposium on Microarchitecture*, Paris, France, 1996, pp. 142–152.

[5] G. H. Loh, D. S. Henry, and A. Krishnamurthy, "Exploiting bias in the hysteresis bit of 2-bit saturating counters in branch predictors," *The Journal of Instruction-Level Parallelism*, vol. 5, June 2003.

[6] M. M. K. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood, "Bandwidth adaptive snooping," in *Proc. 8th Annual International Symposium on High-Performance Computer Architecture*. Cambridge, MA: IEEE Computer Society, 2002, pp. 251–262.

[7] M. Matsumoto and T. Nishimura, "Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, 1998.

[8] A. Moshovos, "RegionScout: Exploiting coarse grain sharing in snoop-based coherence," in *Proc. 32nd International Symposium on Computer Architecture*, Madison, WI, 2005, pp. 234–245.

[9] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," in *Proc. ACM SIGMETRICS*. ACM Press, 2003, pp. 261–271.

[10] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, "Design tradeoffs for the Alpha EV8 conditional branch predictor," in *ISCA '02: Proc. 29th Annual International Symposium on Computer Architecture*. IEEE Computer Society, 2002, pp. 295–306.

[11] J. E. Smith, "A study of branch prediction strategies," in *Proc. 8th Annual International Symposium on Computer Architecture*, 1981, pp. 135–148.