BETTER BRANCH PREDICTION THROUGH PROPHET/CRITIC HYBRIDS

Ayose Falcón
Hewlett-Packard Labs

Jared Stark
Intel

Alex Ramirez
Universitat Politècnica de Catalunya

Konrad Lai
Intel

Mateo Valero
Universitat Politècnica de Catalunya

THE PROPHET/CRITIC HYBRID CONDITIONAL BRANCH PREDICTOR HAS TWO COMPONENT PREDICTORS. THE PROPHET USES A BRANCH'S HISTORY TO PREDICT ITS DIRECTION. WE CALL THIS PREDICTION AND THE ONES FOR BRANCHES FOLLOWING IT THE BRANCH FUTURE. THE CRITIC USES THE BRANCH'S HISTORY AND FUTURE TO CRITIQUE THE PROPHET'S PREDICTION. THE HYBRID COMBINES THE PROPHET'S PREDICTION WITH THE CRITIQUE, EITHER AGREE OR DISAGREE, FORMING THE BRANCH'S OVERALL PREDICTION. RESULTS SHOW THESE HYBRIDS CAN REDUCE MISPREDICTS BY 39 PERCENT AND IMPROVE PROCESSOR PERFORMANCE BY 7.8 PERCENT.

•••••• Processor design is an exercise in trading off performance, power, and energy. Techniques that don't require making this tradeoff, that is, that win in all three metrics, can give your design an advantage over competing designs. Better branch prediction is such a technique. It

- increases performance by reducing the time spent speculating on mispredicted paths,
- reduces power by allowing the processor to run at a lower frequency (hence, lower voltage) and still meet its performance target, and
- reduces energy consumption by reducing the work wasted on misspeculation.

Despite abundant research on branch prediction, researchers have not solved the branch prediction problem—reducing branch mispredicts to the point where they no longer significantly negatively impact processor performance, power, and energy consumption. Leading microarchitects and researchers have said branch prediction will be even more important as pipelines deepen and issue widths increase.

Here, we describe a technique for better branch prediction, which we call *prophet/critic hybrid branch prediction.*[1] We initially describe this technique and the intuition behind why it works by drawing an analogy between running a program and taking a ride in a taxicab. The taxi is the processor, the driver is the branch predictor, and the passenger is the pipeline. The system of roads represents the possible control flow paths through the program. The intersections are branches; that is, points at which the driver must select a particular path to follow. It is the driver's job to navigate the taxi through the system of roads, making the correct turns at intersections, to reach the destination; that is, the end of the program. Wrong turns waste the passenger's time.

A conventional predictor is similar to a taxi with just one driver. The driver takes the passenger to the destination using the knowledge of the roads he acquired from previous trips; that is, using history information stored in the predictor's memory structures. When the driver reaches an intersection, he uses this knowledge to decide which way to turn. The driver accesses this knowledge in the context of the current location. Modern branch predictors[2] access knowledge in the context of the current location (the program counter) plus a history of the most recent decisions that led to the current location.

A prophet/critic hybrid is similar to a taxi with two drivers: one in the front seat and one in the back seat. The front-seat driver has the same role as the driver in the single-driver taxi. This is the *prophet* role. The back-seat driver is the *critic.* The critic watches the turns the prophet makes at intersections. The critic doesn't say anything unless he thinks the prophet has made a wrong turn. When the critic thinks the prophet has made a wrong turn, he waits until the prophet has made a few more turns to be certain they are lost. (Sometimes the prophet makes turns that initially look questionable, but, after a few more turns, in hindsight appear correct.) Only when the critic is certain does he point out the mistake. To recover, they backtrack to the intersection where the critic believes they made a wrong turn, and they try a different direction.

Using prophet/critic hybrids greatly reduces the number of mispredicts. And, as the critic uses more future bits (that is, waits for the prophet to make more turns before reporting they are lost), the reduction in mispredicts grows. Our experiments use some of the best predictors proposed in literature to play the roles of prophet and critic. Depending on the types of prophet and critic we use, a critic using 1 future bit reduces mispredicts by 10 to 20 percent over a prophet scaled up to the same size as the prophet/critic hybrid. For 12 future bits, this reduction grows to 15 to 30 percent.

Related work

Evers and Yeh give a general introduction to dynamic branch prediction.[3] McFarling first proposed hybrid branch predictors.[4] His hybrid has two component predictors and a selection mechanism that decides which to use to predict each branch. Jiménez et al.[5] explain how to arrange two predictors—a small low-latency predictor with poor accuracy and a large high-latency predictor with good accuracy—to provide fast, accurate predictions. In their scheme, the processor initiates a prediction from each predictor in parallel, each prediction being derived from the same stock of branch history information. The small predictor's prediction completes first, and the processor uses it while the other predictor continues its computation. Once computed, this more accurate prediction from the large predictor overrides the earlier prediction if they differ, flushing all work done based on the earlier prediction. Lastly, Grunwald et al.[6] show that using the current branch's prediction in the JRS (Jacobsen, Rotenberg, and Smith) confidence estimator's[7] history register improves speculation control. In our terminology, they use one future bit to obtain a more accurate confidence estimation.

The prophet/critic hybrid combines and builds upon this work; it is a hybrid and uses overriding and future bits. Its key distinction is that, rather than initiating all component predictions at the same time, our method initiates them at different times. This allows the prophet's output to serve as the input to the critic, eliminating the need for a selection mechanism. It also allows the critic to gather multiple (more than one) future bits for a branch. When the branch is on the correct path but mispredicted by the prophet, the critic uses these future bits—of which at least one is wrong because of the mispredict—to train its prediction structures. When the processor encounters the branch again, the critic uses the future bits as context to identify whether the prophet is likely to be wrong and should be overridden. This greatly increases the prediction accuracy for the branch.

Prophet/critic hybrid branch prediction

Current branch predictors make predictions using history information. Once it predicts a

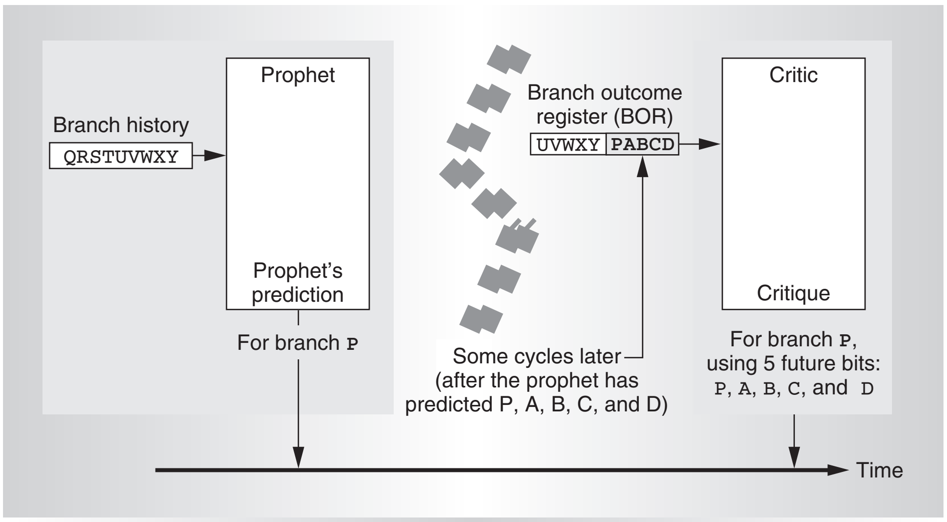
Prophet
Branch history
QRSTUVWXY
Prophet's prediction
For branch P
Branch outcome register (BOR)
UVWXY PABCD
Some cycles later (after the prophet has predicted P, A, B, C, and D)
Critic
Critique
For branch P, using 5 future bits: P, A, B, C, and D
Time

Figure 1. Prophet/critic hybrid structure.

branch, the predictor cannot use information from subsequent predictions to repredict the branch. Our prophet/critic hybrid effectively uses these subsequent predictions. Later, we will describe how to implement our prophet/critic hybrid in a real fetch microarchitecture, and how the prophet and critic can interact without affecting the rest of the processor pipeline. But first, we must describe prophet/critic hybrid branch prediction.

Figure 1 shows the structure of a prophet/critic hybrid. The hybrid consists of two conventional predictors that play the role of either prophet or critic. The prophet provides predictions based on the past behavior of previous dynamic branches (branches Q through Y in the figure). Once it makes a prediction for a branch, the prophet goes on along the predicted path generating new predictions. This prediction (branch P) and the new predictions (branches A through D) form the branch future. As in current history-based predictors, the history helps determine the prediction for the current branch, and that prediction determines the branch's future.

While the prophet generates the branch's future, the critic collects this future information, inserting the prophet predictions into its branch outcome register (BOR). When the critic makes a prediction, its BOR contains two types of branch outcomes:

- outcomes of branches before the one being predicted (branches U through Y), which are branch history, and allow the predictor to correlate on the past, and
- outcomes of the branch being predicted (branch P) and those after it (branches A through D), which are branch future, and allow the predictor to correlate on the future.

Using a combination of past and future correlation, the critic provides a *critique* of each prophet prediction. This critique either agrees or disagrees with the prophet prediction and determines the final prediction for the branch. We use the term *critique* as a synonym for critic prediction, but we tend to use it when we want to stress agreement or disagreement with the prophet. For the remainder of this article, the critic's prediction is the final prediction for the branch.

The prophet/critic hybrid takes advantage of the prophet and critic operating autonomously, predicting the same branch at different times. In a conventional hybrid predictor, both components are accessed in parallel, making predictions for the same branch. A selection mechanism then picks the prediction that is most suitable for the branch. The same situation occurs for overriding predictors. Two predictions begin in parallel on two different predictors. The branch's initial prediction is generated by one predictor in an early pipeline stage, while the final prediction is generated by the other predictor some cycles later. The final prediction overrides the initial

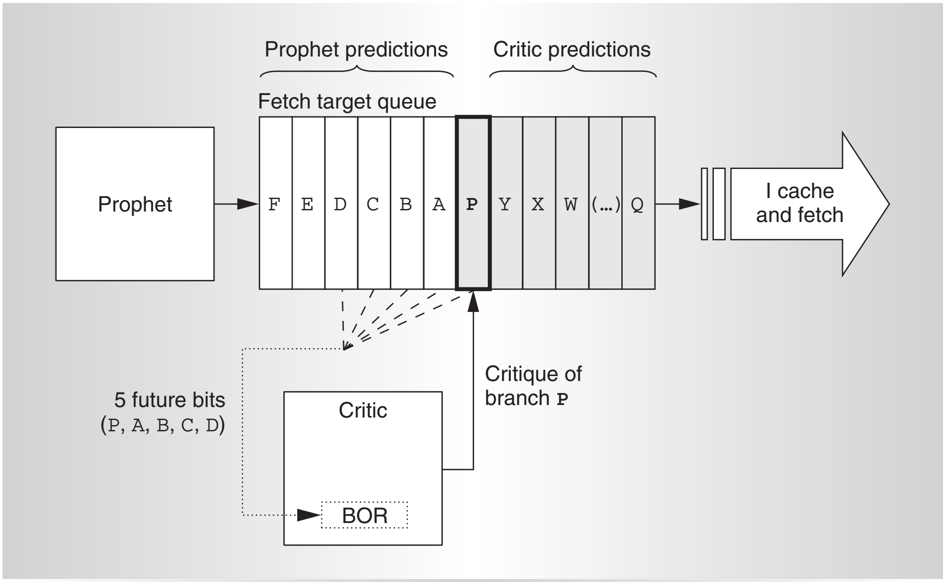

Figure 2. Implementation of a prophet/critic hybrid on a decoupled front-end architecture.

prediction if they differ. In both a conventional hybrid and an overriding predictor, the predictors predict the same branch with the same available information.

However, in the prophet/critic hybrid, although both prophet and critic predict the same branch, they do not initiate their predictions at the same time. Because the critic initiates its prediction some cycles later, it can incorporate information about future code behavior (that is, the branch future provided by the prophet) into its prediction. This future information greatly increases prediction accuracy.

To prevent the delay between the prophet and critic from affecting processor performance, we implement the prophet/critic hybrid on the decoupled front-end architecture[8] shown in Figure 2. A *fetch target queue* (FTQ) decouples the hybrid from the instruction cache. The hybrid produces predictions and inserts them in the FTQ, and the cache later consumes them. Our prophet/critic hybrid requires that the FTQ be full (or mostly full) most of the time. To accomplish this, the hybrid must produce predictions faster than the cache consumes them, so that the FTQ fills to capacity.

The prophet provides the initial prediction for a branch, which it inserts into the FTQ. The cache can immediately consume this prediction, but, since insertions occur at the end of the FTQ and the FTQ is typically full, the prediction usually spends many cycles in the FTQ before the cache consumes it. This and the subsequent predictions that the prophet inserts into the FTQ serve as future bits for the branch, which the critic gathers. When it has gathered the required number of future bits (a fixed number) for the branch, the critic provides a critique of the prophet's prediction. It typically generates the critique well before the cache consumes the prediction.

If the critic agrees with the prophet's prediction, the critic marks the prediction stored in the FTQ as having been criticized, and the critic moves on to the next uncriticized prediction. In Figure 2, unshaded FTQ entries hold uncriticized predictions, and shaded FTQ entries hold predictions that the critic is criticizing or has criticized.

On the other hand, if the critic disagrees with the prophet, the critic takes several actions:

- The critic overrides the prophet's prediction with its own prediction.
- The critic flushes the FTQ entries holding uncriticized predictions.
- The critic redirects the prophet to the path that the critic predicts.

It is important to note that we can confine the flush to the FTQ, and since the cache and the rest of the machine haven't received any

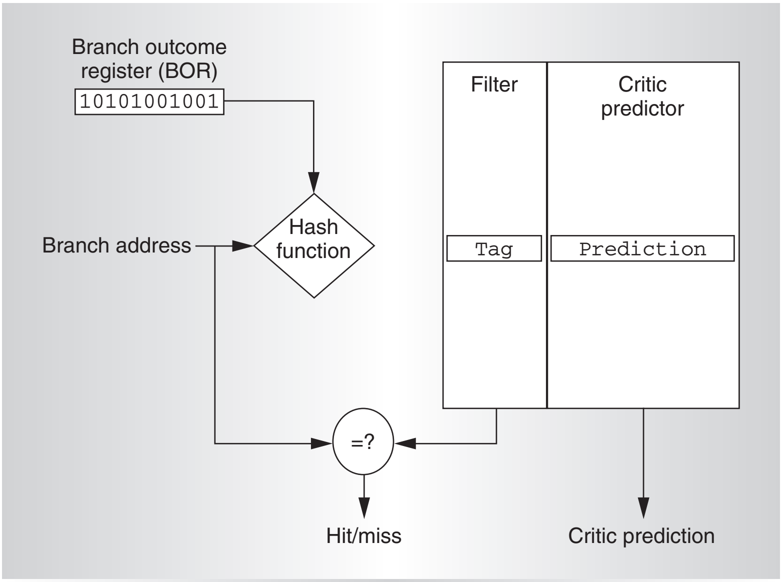

Figure 3. Filtered critic.

of the flushed predictions, they remain unaffected. The criticized predictions are left alone, so if the FTQ is sufficiently full, the flush causes no performance penalty.

Filtering the critic

Multiple branches contending for the same prediction resources—that is, conflicts—can limit the critic's accuracy. We can reduce conflicts by removing or filtering easy-to-predict branches from the critic, allowing the critic to devote all its prediction resources to difficult-to-predict branches.

We have explored filtering the critic. The prophet provides a prediction for every branch, so the processor always has an available prediction regardless of whether the critic provides one. For good performance, the critic should only provide a prediction in the cases where the prophet will likely be wrong. We use prophets that correctly predict 90 to 95 percent of all branches. The critics should then only be responsible for predicting the 5 to 10 percent of branches that the prophets mispredict.

Figure 3 shows a filtered critic, which uses a table of tags to filter branches. When a branch needs a critique, two actions occur in parallel: First, the processor queries the critic for its prediction; and second, the processor accesses the tag table to determine if there is a tag hit. If there is a hit, the critic's prediction serves as the critique. If there is a miss, the critic implicitly agrees with the prophet's prediction, and the processor simply ignores the critic's prediction. The critic is only trained for branches that have hits.

The predictor only inserts new entries into the table when a branch has a tag miss and is mispredicted. The table stores the tag for the particular branch address and BOR value combination so that the next time the predictor encounters that context, it uses the critic's prediction for the branch. The critic's prediction structures are also initialized according to the branch's outcome. We use 8-bit (partial) tags to filter our critics; full tags are unnecessary. The table manages the tags using a least-recently-used (LRU) replacement algorithm.

Simulation methodology

We evaluate our prophet/critic hybrid on a superscalar out-of-order micro-op based microarchitecture derived from the Intel Pentium 4 processor. To reflect where we believe future microarchitectures are headed, this microarchitecture runs at the same frequency as that processor, is twice as wide, has caches that are about twice as big and associative, and has an instruction window (and associated buffers, the scheduling window and load/store buffers) whose size is 16 times that of an Intel Pentium 4 processor.[9,10] In addition, we evaluate our hybrid on a decoupled front-end architecture that uses a decoded instruction cache (storing micro-ops) and so have replaced the original trace-cache-based front-end with the decoupled front-end.

We use an execution-driven IA-32 simulator and 108 benchmarks from a variety of suites: SPECint2000, SPECfp2000, Internet, multimedia, productivity, server, and workstation. In previous work,[1] we fully describe our simulation methodology and benchmarks, and provide expanded simulation results.

Predictors simulated

Any predictor can play the role of prophet or critic. The only restriction is that the critic must be able to use the prophet-generated predictions. We have implemented some of the best predictors proposed in literature and used them as prophet and critic:

- *Gshare.*[4] McFarling found that using global history causes interference in the pattern tables of two-level predictors because branches tend to use a limited number of the possible table entries. His solution is to increase the usefulness of branch history by XORing it together with the branch address. The new indexing mechanism allows branches to share the pattern table in a more efficient way, reducing the aliasing among them.

- *2Bc-gskew.*[11] A derivation of this predictor is in the proposed Compaq Alpha EV8 processor.[12] The original 2Bc-gskew consists of four tables accessed using global history information: a bimodal table (BIM), two gshare-like tables (G0 and G1), and a metapredictor table (META). Depending on the META table's output, the final prediction comes from either the BIM table or the majority vote of the predictions from the BIM, G0, and G1 tables.
- *Perceptron.*[13,14] Perceptron prediction is a two-level scheme using perceptrons instead of two-bit counters as the prediction unit. A *perceptron* is a simple implementation of a neural network that provides predictive capabilities. A perceptron is implemented with a vector of weights, each weight being a signed integer. A branch's address selects a perceptron from a pool of perceptrons. The predictor computes this perceptron's output using global history bits as inputs. The output determines the branch's prediction. If the output is negative, the prediction is not taken; if it is positive, the prediction is taken.

Table 1 shows the parameters we used for simulating the selected predictors. History lengths for gshare and 2Bc-gskew are the maximum usable for their table sizes. For the perceptron predictor, we chose the combination of perceptron table size and history length that gives the highest prediction accuracy.[14]

The table also shows parameters for *tagged gshare*, which is the predictor we used as the critic. It is a filtered version of gshare where each table entry has a tag in addition to the two-bit counter. (Figure 3 shows the structure of a generic filtered critic.) Its structure is like an *N*-way associative cache, with each data item being a two-bit counter. The processor only uses its prediction when there is a tag hit. A tag miss implies implicit agreement with the prophet's prediction. We tuned the number of BOR, history, and future bits that tagged gshare used to achieve the highest possible accuracy.

Table 1. Prophet and critic configurations.

		Total hardware budget	
Predictor	**Parameter**	**8 Kbyte**	**16 Kbyte**
Gshare	No. of table entries	32,768	65,536
	History length (bits)	15	16
Perceptron	No. of perceptrons	282	348
	History length (bits)	28	47
2Bc-gskew	No. of entries per table	8,192	16,384
	History length (bits)	13	14
Tagged gshare	No. of table entries	1024 × 6 way	NA
	BOR size (bits)	18	NA

Simulation results

Our simulation results show the importance of future bits for prediction accuracy, and the impact that the number of future bits has on the critic's performance. Our results also compare the prediction accuracies of prophet/critic hybrids to conventional predictors, and the overall performance of processors with prophet/critic hybrids to those with conventional predictors.

Importance of future bits

Two questions naturally arise regarding future bits: How important are they? And, what is the optimal number needed to provide the best critiques?

Figure 4 shows the mispredict rate (in mispredicts per 1,000 micro-ops) as the number of future bits increases from 0 to 12. Zero future bits means that the predictor uses no future information, which is the same way a conventional hybrid or overriding predictor operates, where all components have the same available history information from which they can generate predictions. We selected five benchmarks (out of 108) showing the different behaviors that we saw when varying the number of future bits. The average line represents the average over all 108 benchmarks. Increasing the number of future bits from 0 to 12 reduces the average mispredict rate by 35

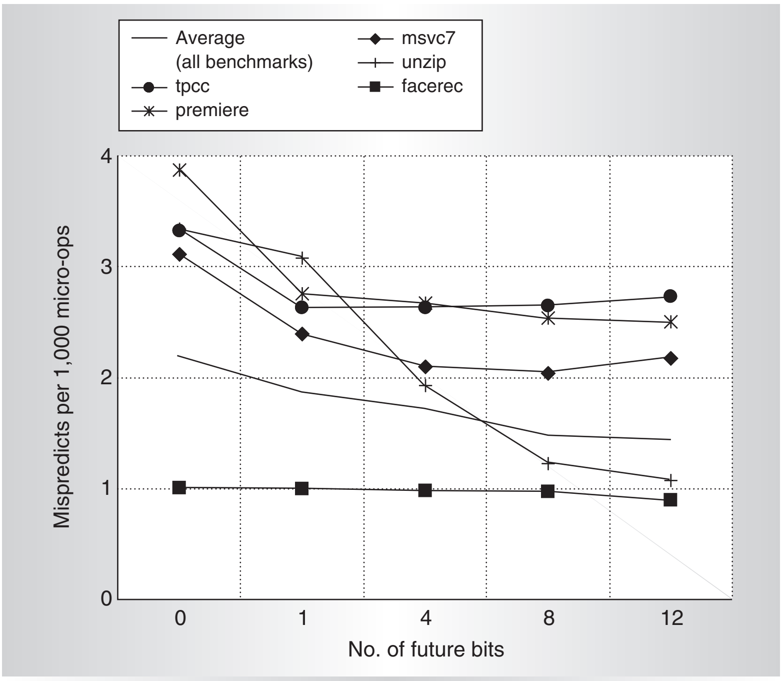

Figure 4. Effect of varying the number of future bits used by the critic on prediction accuracy for selected benchmarks. These results use an 8-Kbyte perceptron for the prophet and an 8-Kbyte tagged gshare for the critic.

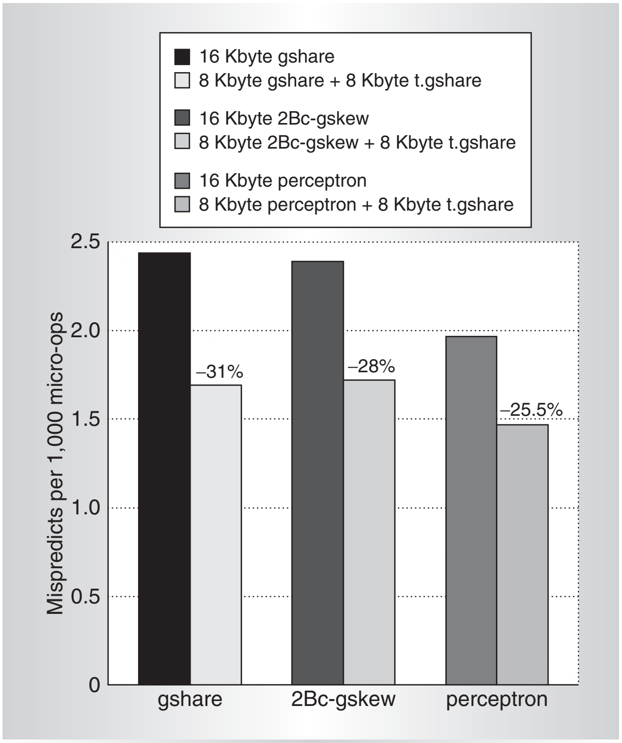

Figure 5. Average mispredict rates of conventional predictors compared to prophet/critic hybrids using eight future bits. Numbers indicate percent reduction in mispredict rate.

percent. Adding just one future bit reduces it by 15 percent. Increasing the number of future bits from one to 12 reduces it by 24 percent (relative to the results for one future bit).

Adding the first future bit reduces the average mispredict rate by 15 percent. The first future bit is the prophet's prediction for the branch. Knowing whether this prediction is taken or not taken is highly valuable information to the critic for determining whether the prophet mispredicted the branch. Our results with other combinations of prophets and critics also show that adding just 1 future bit decreases the mispredict rate as compared to conventional hybrids that don't use future bits.

Increasing the number of future bits beyond one has different effects depending on the benchmark. Both **`tpcc`** and **`facerec`** show little improvement in using more than one future bit, and for **`tpcc`**, there is a small degradation. For **`msvc7`**, adding future bits decreases the mispredict rate up to a point, after which the rate increases. For **`premiere`** and **`unzip`** the mispredict rate continues to decrease as the number of future bits increases.

Thus, adding some future bits always helps, but more is not always better. Research has shown that the optimal number of history bits depends on the static branch[15] and the program's execution phase.[16] This same research is applicable to determining the optimal number of future bits for a prediction.

Prediction accuracy

Figure 5 compares the average mispredict rate of the prophet/critic hybrid to some of the best predictors proposed in literature: gshare, 2Bc-gskew, and perceptron. The prophet was the same as one of those predictors, but used only half the hardware budget. A tagged gshare critic used the other half of the hardware budget. Under these conditions, the results show that a prophet/critic hybrid can reduce the mispredict rate by 25 to 31 percent.

Processor performance

Figure 6 shows average performance measured in micro-ops per cycle. We simulated three prophets (gshare, 2Bc-gskew, and perceptron) combined with a tagged-gshare critic using four, eight, and 12 future bits. The first bar in each group is a prophet alone with the same hardware budget as the prophet/critic hybrids. The results show that as the number of future bits increases, performance also increases. With four future bits, we obtain speedups of 2.7 percent for perceptron, 3.4 percent for 2Bc-gskew, and 4.7 percent for gshare over the performance with a prophet alone. If we use 12 future bits for the critic, speedups grow to 5.2, 7, and 8 percent, respectively.

Why it works—the theory

In 1996, I-Cheng et al.[17] linked data compression to branch prediction.

Compressors typically operate in stream mode, compressing each symbol as the compressor receives it. A predictor generates a probability distribution for the next symbol. When the compressor receives the symbol, it is encoded according to the distribution. The compressor then updates the predictor to generate the distribution for the next symbol. The better the predictor is, the better the compression rate.

Conventional branch predictors and prophets in prophet/critic hybrids are similar to a stream mode compressor's predictor. The branches are the symbols, and they have two possible values: taken and not taken. The predictor handles each branch as it is encountered. The prediction is the symbol (branch direction) with the highest probability. After it makes the prediction, the predictor is updated. Operating in stream mode limits the predictor to using previously encountered branches (branch history) to generate the probability distribution. This limits its accuracy.

Critics, on the other hand, are different. They do not operate in stream mode. When critics encounter a branch, they wait until they have a few following branches before they provide a prediction. Because they wait, they can use a probability model (such as a Markov model) that generates a probability distribution for the branch using the outcomes of the branches both before and after the branch in question; that is, they use both a branch's history and future. Since critics use a predicted branch future instead of the actual future, they actually maintain a probability model for whether the prophet's prediction is wrong, rather than a model for the branch's outcome. However, it is easy to generate the predicted outcome from the prophet's prediction plus the critic's prediction of whether the prophet's prediction is wrong. Because critics do not operate in stream mode, they can delay making their predictions, greatly reducing the number of mispredicts.

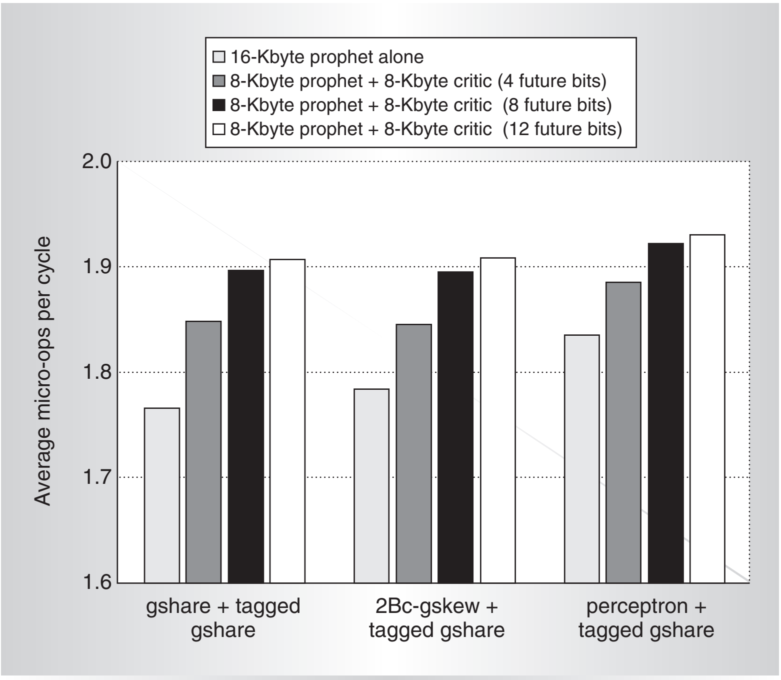

Figure 6. Average performance for 16-Kbyte conventional predictors compared to an 8-Kbyte prophet plus an 8 Kbyte critic, using four, eight, and 12 future bits.

Our results show that an 8-Kbyte perceptron prophet with an 8-Kbyte tagged gshare critic has 39 percent fewer mispredicts than a 16-Kbyte 2Bc-gskew predictor—a predictor similar to that in the proposed Compaq Alpha EV8 processor—across a wide range of applications. For `gcc`, the percentage of mispredicted branches drops from 3.11 to 1.23 percent. The number of micro-ops between pipeline flushes because of mispredicts limits the amount of parallelism that we can extract from the instruction window, and hence bounds the useful size of the window. The prophet/critic hybrid increases this number from 418 for the 2Bc-gskew predictor to 680. On a machine based on the Intel Pentium 4 processor, this improves micro-ops-per-cycle performance by 7.8 percent (18 percent for gcc). In addition, it reduces the number of micro-ops processed along both correct and incorrect paths—a measure of the energy required to accomplish a task—by 8.6 percent.

Acknowledgments

We thank the members of the Microarchitecture Research Lab for their help and comments on this work, and the folks of Intel Labs Barcelona for providing us with the help and infrastructure needed to finish this work.

This work was supported by the Ministry of Science and Technology of Spain under contracts TIC–2001–0995–C02–01 and TIN–2004–07739–C02–01, by the HiPEAC European Network of Excellence, and by an internship provided by Intel.

References

1. A. Falcón et al., "Prophet/Critic Hybrid Branch Prediction," *Proc. 31st Int'l Symp. Computer Architecture* (ISCA 31), IEEE CS Press, 2004, pp. 250-262.
2. T.-Y. Yeh and Y.N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," *Proc. 19th Int'l Symp. Computer Architecture* (ISCA 19), ACM Press, 1992, pp. 124-134.
3. M. Evers and T.-Y. Yeh, "Understanding Branches and Designing Branch Predictors for High-Performance Microprocessors," *Proc. IEEE*, vol. 89, no. 11, Nov. 2001, pp. 1610-1620.
4. S. McFarling, *Combining Branch Predictors,* tech. report TN-36, Compaq Western Research Lab., 1993.
5. D.A. Jiménez, S.W. Keckler, and C. Lin, "The Impact of Delay on the Design of Branch Predictors," *Proc. 33rd Int'l Symp. Microarchitecture* (Micro-33), IEEE CS Press, 2000, pp. 67-76.
6. D. Grunwald et al., "Confidence Estimation for Speculation Control," *Proc. 25th Int'l Symp. Computer Architecture* (ISCA 25), IEEE CS Press, 1998, pp. 122-131.
7. E. Jacobsen, E. Rotenberg, and J.E. Smith, "Assigning Confidence to Conditional Branch Predictions," *Proc. 29th Int'l Symp. Microarchitecture* (Micro-29), IEEE CS Press, 1996, pp. 142-152.
8. G. Reinman, T. Austin, and B. Calder, "A Scalable Front-end Architecture for Fast Instruction Delivery," *Proc. 26th Int'l Symp. Computer Architecture* (ISCA 26), IEEE CS Press, 1999, pp. 234-245.
9. H. Akkary, R. Rajwar, and S.T. Srinivasan, "Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors," *Proc. 35th Ann. Int'l Symp. Microarchitecture* (Micro-35), IEEE CS Press, 2003, pp. 423-434.
10. A. Cristal et al., "Out-of-order commit processors," *Proc. 10th Int'l Conf. High Performance Computer Architecture* (HPCA-10), IEEE CS Press, 2004, pp. 48-59.
11. A. Seznec and P. Michaud, *Dealiased Hybrid Branch Predictors*, tech. report PI-1229, Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), 1999.
12. A. Seznec et al., "Design Tradeoffs for the EV8 Conditional Branch Predictor," *Proc. 29th Int'l Symp. Computer Architecture* (ISCA 29), IEEE CS Press, 2002, pp. 295-306.
13. L.N. Vintan and M. Iridon, "Towards a High-Performance Neural Branch Predictor," *Proc. Int'l Joint Conf. Neural Networks* (IJCNN 99), IEEE Press, 1999, pp. 868-873.
14. D.A. Jiménez and C. Lin, "Neural Methods for Dynamic Branch Prediction," *ACM Trans. Computer Systems,* vol. 20, no. 4, Nov. 2002, pp. 369-397.
15. J. Stark, M. Evers, and Y.N. Patt, "Variable Length Path Branch Prediction," *Proc. 8th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS VIII), ACM Press, 1998, pp. 170-179.
16. T. Juan, S. Sanjeevan, and J.J. Navarro, "Dynamic History-Length Fitting: A Third Level of Adaptivity for Branch Prediction," *Proc. 25th Int'l Symp. Computer Architecture* (ISCA 25), IEEE CS Press, 1998, pp. 155-166.
17. I.-Cheng et al., "Analysis of Branch Prediction Via Data Compression," *Proc. 7th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS VII), ACM Press, 1996, pp. 128-137.

Ayose Falcón is a research scientist at Hewlett-Packard Labs in Barcelona. His research interests include fetch unit optimizations, especially branch prediction and instruction prefetching, and microprocessor simulation. Falcón has a BS and an MS in computer science from the University of Las Palmas de Gran Canaria, and a PhD in computer science from the Uni-

versitat Politècnica de Catalunya (UPC). He is a member of the ACM.

Jared Stark is a research scientist at Intel's Microarchitecture Research Lab in Hillsboro, Oregon. His research interests include branch prediction, dynamic instruction scheduling, and aggressive speculation. Stark has a BS in electrical engineering and an MS and a PhD in computer engineering, all from the University of Michigan. He is a member of the IEEE.

Alex Ramirez is an assistant professor in the Computer Architecture Department at UPC. His research interests include profile-guided compiler optimizations, code layout optimizations, performance studies of user and system code like database applications, and the design and implementation of the fetch stage of superscalar and multithreaded processors. Ramirez has a BS, an MS, and a PhD in computer science from UPC. He is a member of the ACM.

Konrad Lai is a senior principal research scientist at Intel's Microprocessor Technology Laboratories, Hillsboro, Oregon. His research interests include microprocessor, memory, and system architectures. Lai has a BS in electrical engineering from Princeton University and an MS in computer engineering from Carnegie Mellon University. He is a member of the ACM and the IEEE.

Mateo Valero is a professor in the Computer Architecture Department at UPC. His research interests include high-performance computer architectures. Valero has a BS in telecommunication engineering from the Polytechnic University of Madrid and a PhD in telecommunications from UPC. He is a Fellow of the IEEE and the ACM.

Direct questions and comments about this article to Ayose Falcón, Hewlett-Packard, Avda. Graells 501, 08714, Sant Cugat del Valles, Barcelona, Spain; ayose.falcon@hp.com.