Assigning Confidence to Conditional Branch Predictions

Erik Jacobsen, Eric Rotenberg†, and J. E. Smith
Departments of Electrical and Computer Engineering and
† Computer Sciences
University of Wisconsin-Madison
Madison, WI 53706
jacobsen@cae.wisc.edu, ericro@cs.wisc.edu, jes@ece.wisc.edu

Assigning Confidence to Conditional Branch Predictions

Erik Jacobsen, Eric Rotenberg†, and J. E. Smith
Departments of Electrical and Computer Engineering and
† Computer Sciences
University of Wisconsin-Madison
Madison, WI 53706
jacobsen@cae.wisc.edu, ericro@cs.wisc.edu, jes@ece.wisc.edu

Abstract

Many high performance processors predict conditional branches and consume processor resources based on the prediction. In some situations, resource allocation can be better optimized if a confidence level is assigned to a branch prediction; i.e. if the quantity of resources allocated is a function of the confidence level. To support such optimizations, we consider hardware mechanisms that partition conditional branch predictions into two sets: those which are accurate a relatively high percentage of the time, and those which are accurate a relatively low percentage of the time. The objective is to concentrate as many of the mispredictions as practical into a relatively small set of low confidence dynamic branches.

We first study an ideal method that profiles branch predictions and sorts static branches into high and low confidence sets, depending on the accuracy with which they are dynamically predicted. We find that about 63 percent of the mispredictions can be localized to a set of static branches that account for 20 percent of the dynamic branches. We then study idealized dynamic confidence methods using both one and two levels of branch correctness history. We find that the single level method performs at least as well as the more complex two level method and is able to isolate 89 percent of the mispredictions into a set containing 20 percent of the dynamic branches. Finally, we study practical, less expensive implementations and find that they achieve most of the performance of the idealized methods.

1. Introduction

It is becoming common practice in high performance processors to predict conditional branches [4, 7, 9, 13] and speculatively execute instructions based on the prediction [2, 8]. Typically, when speculation is used, all branch predictions are acted upon because there is low penalty for speculating incorrectly. I.e. most resources available to speculative instructions would be unused anyway. And, on average, a branch prediction will be correct a high percentage of the time.

However, as processors become more advanced, we can envision implementations where the penalty for an incorrect speculation may be high enough that it may be better not to speculate in those instances where the likelihood of a branch misprediction is relatively high. That is, it may be desirable to vary behavior depending on the likelihood of a misprediction. Consequently, we would like to develop hardware methods for assessing the likelihood that a conditional branch prediction is correct; we refer to these as *branch prediction confidence mechanisms*. Consider the following potential applications.

1) Selective Dual Path Execution: Resources may be made available for simultaneously executing instructions down both paths following a conditional branch. However, it will likely be too expensive to follow both paths after all branches, especially when several conditional branches may be unresolved at any given time. Consequently, it may be desirable to set a limit of two threads at any given time and to fork a second execution thread for the non-predicted path only in those instances when a branch prediction is made with relatively low confidence. After most predicted branches only the predicted path would be speculatively followed, but occasionally, both paths would be followed.

2) Guiding instruction fetching in simultaneous multithreading (SMT): In SMT, instruction fetching has been identified as a critical resource [10]. This resource can be more efficiently used by fetching instructions only down predicted paths that have a high likelihood of being correctly predicted. That is, threads predicted with a high confidence should be given priority over those with low confidence. This will reduce the number of wasted instruction fetches caused by following the wrong speculative path.

3) Dynamic Selector for a hybrid branch predictor: Hybrid branch predictors [1, 5] use more than one predictor and select the prediction made by one of them based on the history of prediction accuracies of the constituent predictors. The methods proposed in [1, 5] are basically *ad hoc* confidence mechanisms developed for this specific application. By studying confidence mechanisms in general, we may be able to arrive at more accurate hybrid

selectors.

4) Branch Prediction Reverser: If the confidence in a branch prediction can be determined to be less than 50%, then the prediction should be reversed. Hence, a confidence mechanism could be used to generate a "reverse prediction" signal for those branch predictions with a less than 50% accuracy.

In theory, one could focus on developing hardware that computes probabilities that individual branch predictions are correct (or incorrect). However, in practice this could be rather complex (because a division is implied), and a computed probability for each branch is not what is really wanted, anyway. Rather, we attempt to divide branch predictions into two sets: those in which there is high confidence, and those in which there is low confidence. A binary signal is generated simultaneously with a branch prediction to indicate the confidence set to which the prediction belongs. The pair of prediction and confidence signals are illustrated in Fig. 1. To see how a confidence signal could be used, consider again the four applications listed above.

1) For Selective Dual Path Execution, the confidence signal can be used to trigger the forking of a second thread for low confidence branch predictions.

2) For the SMT instruction fetching application, the confidence signal can be used to enable instruction fetching for speculative threads in which there is high confidence.

3) For designing a hybrid prediction selector, confidence signals from the multiple predictors can be compared to select the prediction to be used.

4) For the reverser application, if the confidence threshold can be set at approximately 50% accuracy, then the confidence signal can be used to reverse a prediction.

For each of these applications, we would like to divide the predictions into high and low confidence sets and concentrate as many of the mispredicted branches as possible into the low confidence set -- while at the same time keeping the low confidence set relatively small. Note that in general, one could divide the branches into multiple sets with a range of confidence levels. To date, we have not pursued this generalization and consider only two confidence sets in this paper.

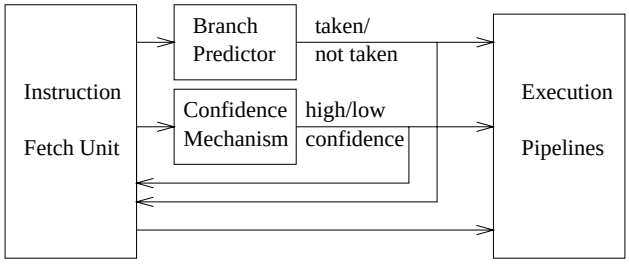

Fig. 1. A generic speculative processor containing a branch prediction signal paired with high/low confidence signal.

To generate the signal that separates the two confidence sets, we propose using benchmarks to collect prediction accuracy data. This data can then be used to design logic so that the high and low confidence sets have the characteristics we desire. Note that this logic is designed using data *for our selected benchmarks*. However, once implemented, the confidence logic is used for all programs. That is, to simplify the hardware design, we do not dynamically adjust the criteria for determining the high and low confidence sets.

1.1. Previous Work

In [9] there is a proposal for assigning confidence levels to different counter values in predictors based on saturating counters. There is also a relatively abstract example of optimizing performance by speculating to different degrees based on the confidence level.

In [12] the authors use branch probability levels to guide disjoint eager execution when forking multiple threads. Multiple threads are forked, with the most probable being forked first. Because of the difficulties with dynamically computing the probabilities, static profile-based probabilities are used in the suggested implementation.

Regarding the specific application to reversing predictions, some processors have static "prediction reversal" bits. The Livermore S-1 [3] made a static branch prediction, but had a dynamic "reverse" bit in the instruction cache that was used to reverse the static prediction after it was found to be incorrect. The more recent PowerPC 601 microprocessor [6] makes a prediction based on the opcode and direction of the branch, but allows the compiler to place a "reverse" bit in the instruction to change the default prediction.

1.2. Simulation Methodology

We collected branch prediction accuracy data using trace-driven simulation. For benchmarks, we used the Mach version of the IBS benchmark suite [11] -- chosen because they more accurately represent branch characteristics of real programs than the commonly used SPEC benchmarks, and, because they include kernel code.

We arrive at composite data for the collection of benchmarks by averaging. We do this by weighting the results so that each benchmark, in effect, executes the same number of conditional branches.

An important part of the study is the underlying branch predictor. In most of our simulations, we use a fairly aggressive predictor. It is the gshare predictor [5] with 2^{16} entries -- each of which is a saturating 2-bit

counter. The counter array is addressed with the exclusive-OR of bits 17 through 2 of the program counter and the most recent 16 branch outcomes held in a branch history register (BHR). In Section 5 we consider performance using less expensive predictors.

1.3. Paper Overview

We focus on methods of assigning confidence to branch predictions -- not the applications of the confidence methods. We are currently investigating some of the more interesting applications, and our goal here is to establish a set of base confidence methods to use as a starting point for other studies. In Section 2, we collect branch prediction accuracy statistics and relate them to static branches. This exercise establishes the general method we will use to display confidence results, and it suggests an optimal static confidence method that we use as a baseline for comparing the dynamic methods to follow. In Section 3, we look at a number of general dynamic confidence methods, using both one and two levels of tables. In Section 4, we give some experimental results for the dynamic methods using the IBS benchmark suite. In Section 5, we consider some practical implementation issues for dynamic confidence methods with a goal of reducing logic complexity and cost. Section 6 concludes the paper.

2. Analysis of Static Branches

We first consider an idealized method where all static branches are assigned either high or low confidence levels, based on the accuracy with which they can be predicted. We begin with the relatively powerful gshare dynamic predictor described in Section 1.2 and collect statistics for each static branch: 1) the number of times the branch is executed, and 2) the number of incorrect predictions. Consequently, the misprediction rate for each static branch can be generated.

Then we combine the branches for all the benchmarks and normalize them so that each benchmark effectively contributes the same number of dynamic branches. Next, we sort the static branches according to their misprediction rates, highest rate first. This concentrates the most difficult to predict static branches at the top of the sorted list, and the sorted list can then be used to divide the predicted branches into low and high confidence sets.

Of course, many such sets are possible, depending on what we define to be "low" and "high" -- we have thus far been vague on this point. To understand the range of possibilities, we go down the the sorted list of branch statistics and plot accumulated fractions of mispredicted branches versus the accumulated fractions of executed branches that produce them. For the IBS benchmarks, Fig. 2 is the resulting plot.

Note that the graph has discrete data points (although they run together when plotted), corresponding

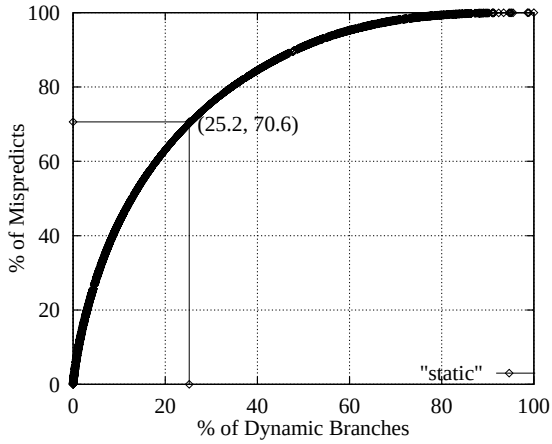

Fig. 2. Cumulative mispredictions versus cumulative dynamic branches.

to each static branch that occurs in the sorted list. Each such point defines a pair of high and low confidence branch prediction sets. To interpret the graph, consider the data point (25.2,70.6), marked on the graph. At this point, 25.2 percent of the dynamic branches have been accumulated by the time we reach the static branch in question, and 70.6 of the mispredictions are included. That is, we can separate the branches so that 25.2 percent of executed branches are placed in the low confidence set and account for 70.6 percent of the mispredictions.

The general shape of the curve is a steep rise that rounds a "knee" into a nearly horizontal line. This particular curve for static branches has a rather gentle knee; some of the dynamic methods given in the next section have much sharper knees. The steeper the initial slope and the farther to the left the knee occurs, the better. With a curve of this shape, more mispredictions are concentrated within a smaller set of low confidence predictions. Or, in other words, the branches in the lower confidence set have a higher misprediction rate, and, conversely, the branches in the higher confidence set have a lower misprediction rate. The ideal low confidence set would consist solely of mispredicted branches. The corresponding graph would have a straight line parallel to the y-axis shifted to the right from the origin by the mispredict rate.

One could develop a static branch confidence method based on the procedure just followed. That is, one could profile branches as we have just done. Then a threshold misprediction rate could be set, with all static branches above the threshold being tagged one way (low confidence), and those below tagged another (high confidence). Or, alternatively, a fraction of mispredictions could be chosen, and the corresponding set of static branches could be selected as the low confidence ones.

For the static confidence method just described, the graph in Fig. 2 provides an optimistic estimate of the performance. The method is optimistic because it represents "perfect" profiling -- we are executing the programs with exactly the same data as for the profile. Nevertheless, we use the results from this optimistic static method for comparison with the dynamic methods we consider in the following sections. At a minimum, we would like the dynamic methods to exceed the performance of the static method.

3. Dynamic Confidence Mechanisms

In the previous section we partitioned static branches into high and low confidence sets. That is, all dynamic predictions of the same static branch are assigned to the same confidence set. However, we can also partition branches so that dynamic predictions of the same static branch can be assigned to different confidence sets. This partitioning is done based on dynamic history, and we refer to these as *dynamic confidence mechanisms*. There are a large number of dynamic confidence mechanisms available. They are first cousins of dynamic branch predictors, and many such branch predictors have been proposed over the years [4, 7, 9, 13]. In this paper, we cannot explore the entire design space. But we select some representative confidence methods -- those that preliminary experiments indicated as being more interesting variations.

We begin with generic confidence mechanisms that are somewhat idealized. In Section 5 we refine them to more practical implementations.

3.1. One-Level Confidence Methods

The one level dynamic confidence methods are so-named because they use a single level of table lookup; this is illustrated in Fig. 3. The table contains the *n* most recent correct/incorrect indications for the given table entry. These n-bit entries are essentially shift registers. We call each of these a Correct/Incorrect Register (CIR, pronounced "sir"), and we refer to the entire table as the CIR Table (CT). We use the convention that a 1 in a CIR indicates an incorrect prediction, and a 0 indicates correct prediction. For example if a prediction is correct 3 times, followed by an incorrect prediction, followed by 4 correct predictions, then an 8-bit CIR contains 00010000.

There are a number of possibilities for accessing the table. In one, the (truncated) program counter for a branch instruction is used as an index into the CT. Alternatively, one could keep track of global branch outcomes in a branch history register (BHR) and use it to index into the CT. A third alternative is to maintain a global CIR (i.e. correct/incorrect status collected for the most recent dynamic branches) and use it to index into the CT.

Beginning with these three basic methods of indexing into the CT (PC, global BHR, global CIR), one can construct a number of others by concatenating portions of

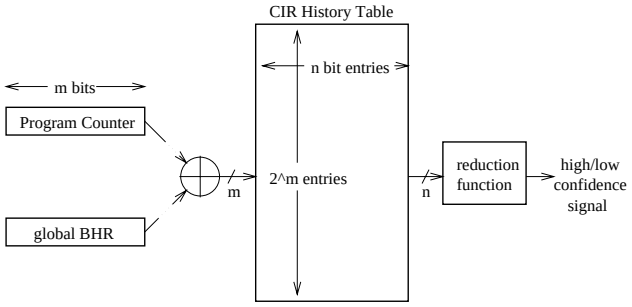

Fig. 3. One Level Dynamic Confidence Mechanism(s).

each or exclusive-ORing them. Preliminary studies we have done indicate that exclusive-ORing is more effective than concatenating sub-fields. However, this can be influenced by the table size and deserves further study. Our preliminary studies also indicate that indexing with a global CIR is of little value -- it gives low performance when used alone and typically reduces performance when added to the others. Consequently, we will report results only for implementations using the PC and global BHR. This leads to three variations for indexing into the CT: PC alone, global BHR alone, and PC xor BHR.

Now, to divide the branches into two sets, we have to take the CIR accessed from the table and reduce it to a single binary signal. In general, we do this by passing the CIR through a combinational logic block, named the "reduction" function in Fig. 3. For example, the reduction function could perform a ones count on the CIR: more ones indicate a higher number of recent mispredictions which would tend to indicate lower accuracy. In Section 5, we will consider this ones count reduction function, along with some others.

In an actual implementation, complete CIRs could be kept in the CT, with a separate logic block implementing the reduction function, exactly as shown in Fig. 3. However, one might also use implementations that maintain a compressed form of the CIRs in the CT along with a simplified reduction function. These will be discussed more in Section 5.

3.2. Two Level methods

With two level dynamic confidence methods, we index into a first level CT in a manner similar to the one level methods, then combine the CIR read from the table with some combination of the PC and global BHR to index into a second level CT. The second level table contains the Correct/Incorrect values for the *p* most recent times the first level CIR/PC/BHR combination occurred. Finally, the CIR read out of the second level table passes through a combinational reduction function as in the previous one level methods.

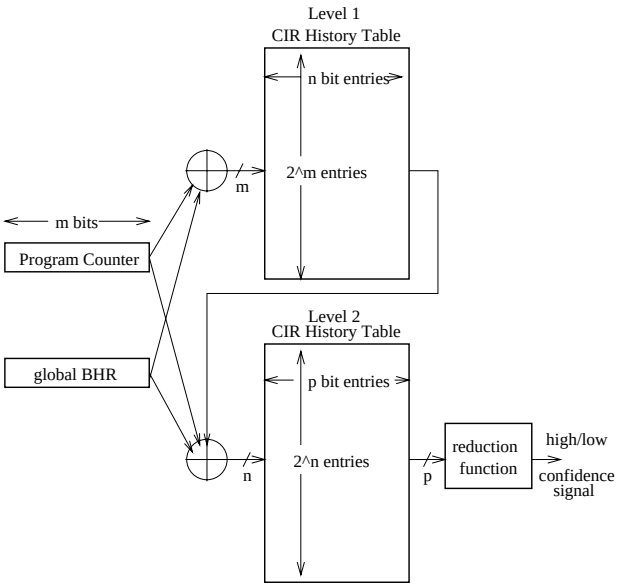

Fig. 4. Two Level Dynamic Confidence Mechanism(s).

There are several variations of the two-level method (refer to Fig. 4). In general, one can hash some combination of PC and global BHR for the first table, then hash the output of the first table with some combination of PC and global BHR. This leads to 12 different possibilities -- and other two level structures could probably be made (one could consider using the global CIR when computing the index into the second level table, for example). After some preliminary exploration, we settle on only three representative methods.

In the first variation, the PC alone is used to read the first table, and the CIR alone is used to access the second level table. In the second variation, the PC xor BHR is used to read the first level table, and the CIR read from the table is used to read the second level table. The third variation is like the second except the PC and BHR are exclusive-ORed with the CIR read from the first level table before indexing into the second level table.

4. Experimental Results

We now use trace-driven simulation to study the dynamic confidence methods outlined in the previous section. As stated earlier, the underlying branch predictor is a gshare predictor using a table with 2^{16} two-bit counters. The prediction table is accessed with the exclusive-OR of the 16 low order PC bits and a 16 bit global BHR. For the one level confidence methods, the CIR tables also have 2^{16} entries, each of which contains a 16 bit CIR. We simulate the IBS benchmark suite. Results are averaged by weighting the individual benchmarks so that each contributes the same total number of dynamic branches. For the relatively large underlying branch predictor we use, the overall misprediction rate is 3.85 percent.

For all methods, we initialize the branch predictor table to "weakly taken" and the CIR tables to all ones. All ones was found to give better results than other initial CT values we studied; additional information on initial values will be given in Section 5.

Initially, we will collect separate statistics for each CIR value read from the CT (the second level CT in the case of the two level methods). For each CIR pattern we keep track of number of times the pattern appears and the fraction of incorrect predictions that occurred when that CIR pattern was read.

After collecting this data for each CIR pattern, we sort and construct a graph similar to that used for the static method in section 2. In particular, we sort the CIR patterns according to misprediction rates, highest rate at the top. The sorted list is used to plot data points -- one per CIR. For each point the X axis value is the fraction of accumulated conditional branches corresponding to this CIR and those higher on the list; the Y axis value is the fraction of mispredictions the conditional branches account for. Now, each point can be used to define high and low confidence prediction sets. A combinational reduction function that detects the sets selects only those CIRs above (low confidence) or below (high confidence) the CIR in question. The CIRs define minterms for the reduction function.

As in the static method, this method of determining the confidence sets is idealized because the reduction function is tuned to a specific set of data input values. In addition, the combinational reduction function could be very complicated, i.e. it could have many prime implicants -- many of which could conceivably be minterms. In Section 5 we look at more practical reduction functions.

4.1. One Level Methods

Fig. 5 gives performance graphs for one level methods: indexing with PC alone, global BHR alone, and PC xor BHR. Note that in order to avoid the data points running together (as they do in Fig. 2) we only plot those points that differ from a previous point by 2.5 percent. The best method (the one that concentrates the largest set of mispredictions into the smallest set of predictions) uses PC xor BHR to index into the table. The reason is that the PC and the BHR together more precisely establish a context for the branch in question than either alone (essentially the same reasoning that leads to the gshare predictor). A close second in performance indexes with the global BHR alone, and the worst performance is provided by using the PC alone. Consider the points in the curves that correspond to a low confidence set containing 20 percent of all the branch predictions (20 percent is chosen rather arbitrarily for illustrative purposes). Indexing with PC xor BHR concentrates 89 percent of the mispredictions

into the low confidence set; BHR alone concentrates 85 percent, and the PC concentrates 72 percent.

Also shown in the Fig. 5 graph is a curve for the static method given in Section 2. We see that the dynamic methods are capable of performing much better than the optimistic static method. For comparison, with the static method 20 percent of the branches concentrate only about 63 percent of the mispredictions.

For the dynamic methods, the all zeros CIR occurs frequently, and we refer to this all zeros CT entry as the "zero bucket". The zero bucket corresponds to the case where the table entry has seen a correct prediction the last 16 times in a row. It is not surprising that the zero bucket is accessed frequently, given that the overall prediction accuracy is 96.15 percent. The large zero bucket explains the long gap between data points for the dynamic methods in the right side of the graph. For example, with the two better dynamic methods, about 80 percent of the branch predictions lead to the all zeros CIR, and 12-15 percent of the mispredictions occur with the all zeros CIR. Hence, in the 20 to 100 percent dynamic branch region of the graph, the dynamic methods have no data points. The static method does have data points, however, and these points allow the static curve to arc above the interpolated curves for the dynamic methods in this region.

Finally, we note once again that the dynamic results are idealized in a way similar to the way the static branches are. In particular, we sort the CIRs from worst to best based on their performance for the IBS benchmarks, and effectively use an optimal reduction function for the resulting CT. When we look at practical reduction functions, this level of optimism will be mitigated.

4.2. Two Level Methods

Now, we consider the two level confidence methods. The results are shown in Fig. 6 The best method accesses the first level table with PC xor BHR, and the second level table is accessed only with the CIR read from the first level table. The method that accesses the first level table with PC xor BHR and the second level table with the first level CIR xor PC xor BHR is generally the second best performer. However, there is a region in the 5 to 10 percent range (X-axis) where the third method -- PC accessing the first level, CIR the second level -- is slightly better. Otherwise the third method is worse than the other two level methods we chose to simulate. As with the one level methods, the all zeros CT entry is very large and leads to no data points from the 10-30 percent region to the 100 percent point on the X-axis.

In Fig. 7, we compare our best one level method (from Fig. 5) with our best two level method (from Fig. 6) and the static method (from Fig. 2). We see that the one and two level methods give very similar performance. If anything, the two level method performs very slightly worse. The main difference is the presence or absence of the second level table; the first level tables are identical. Consequently we conclude that the extra hardware in the second level table is not worth the cost, at least when we use ideal reduction functions. Additional studies with realistic reduction functions tend to reinforce this conclusion, so we do not consider two level tables further in this paper.

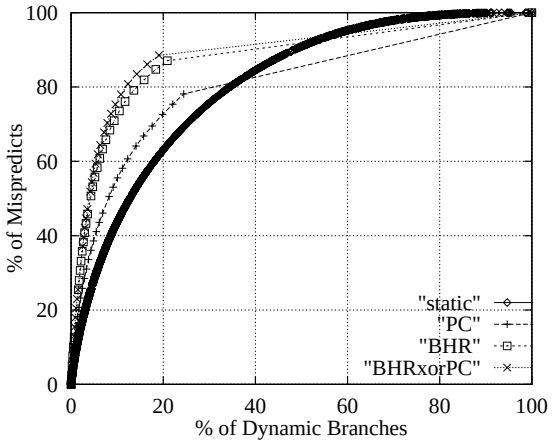

Fig. 5. Cumulative mispredictions versus cumulative dynamic branches for one level dynamic confidence methods.

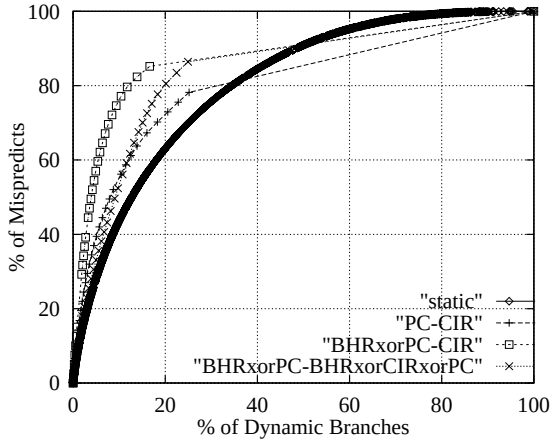

Fig. 6. Cumulative mispredictions versus cumulative dynamic branches for two level dynamic confidence methods.

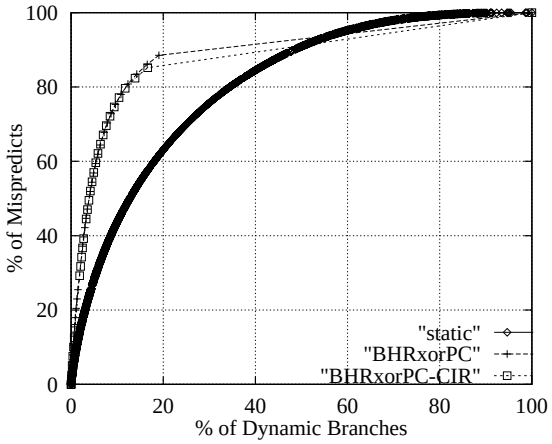

Fig. 7. Comparison of best one level, two level, and static methods.

5. Implementation Issues

In this section we apply the results from the previous section to arrive at actual implementations. First, we look at reduction functions, then consider what we call the "granularity" issue, cost considerations, and initialization issues.

5.1. Reduction Functions

We now propose some simple reduction functions and compare results they provide with the relatively optimistic results from the previous section. For the obvious reasons of reducing cost and/or logic complexity, we would like simple reduction functions. We have arrived at three by inspection of the sorted lists of CIRs, guided by our intuition.

We have already observed that PC xor BHR indexing performs best of the one level methods we studied. Consequently, in this section, we focus on this method. Note that we will occasionally refer to this method simply as the "best", even though it is only the best *of those we studied*.

Ones Counting. First, we consider counting the ones in the CIRs read from the table. The reasoning is that the more ones in the CIR, the higher the number of recent mispredictions, and therefore the more likely there will be a future mispredict. Because the CIRs are length 16 in the study given above, we have 17 data points (0 ones, i.e. no mispredictions out of the last 16, a single one, i.e. one mispredict, etc. up to 16 ones). The results for ones counting and for the other reduction methods considered in this section are plotted in Fig. 8. On the same graph is the plot for the optimistic reduction function we are trying to approximate; the optimistic curve does not have the discrete data points plotted to make the other points easier to read. Note that for ones counting the zero bucket lines up with the optimistic zero bucket (as it should). The one level method with ones counting falls short of optimum for other data points, however.

The reason is that with a one level table, there is significant time ordering in the CIR, with the most recent 16 predictions being represented, in time sequence. However, recent mispredictions, e.g. the most recent, correlate better than the older ones, e.g. 16 predictions ago. Yet, with ones counting, they are all given equal weight.

Saturating Counters. A second reduction function that we consider is a saturating ones count on the CIR values. We do not expect this method to perform any better than ones counting. However, it could lead to a less costly confidence method because the saturating counters can be embedded into the CT in place of the CIRs -- leading to an essentially logarithmic reduction in table space. Using counters that count from 0 to 16 allow us to compare directly with the ones counters from the preceding subsection (although a slightly less expensive implementation would count from 0 to 15). We count up for each correct prediction and down for each incorrect one, saturating at the extremes of 0 and 16. We once again have 17 data points which we plot in Fig. 8. We see that saturating counters have a potential deficiency. In particular, the equivalent of the zero bucket, the maximum saturated counter, becomes significantly larger; i.e. it contains more mispredicted branches. This happens because branches that are correctly predicted a vast majority of the time usually access a maximum saturated counter. If there is a single misprediction, followed by a correct prediction, then the saturating count will be non-maximum for only one branch. On the other hand, if the full CIR is kept and a ones count is used, the single misprediction will lead to

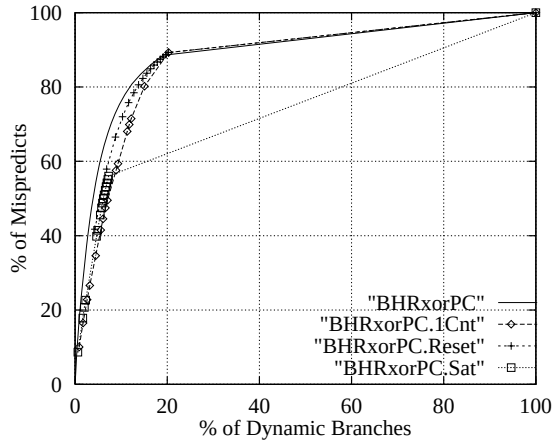

Fig. 8. Confidence graph for one level method with reduction via ones counting, saturating counting, and resetting counting.

a nonzero CIR for the next 16 branches. This means many non-zero CIRs containing a single one are "converted" to the maximum saturated count value.

For the portions of the curves at the extreme left of the graph, i.e. before the zero bucket region is reached, saturating counters behave about as well as ones counting. Consequently, if we choose to partition the confidence sets in this region, i.e. with 10 percent or fewer of the dynamic branches in a low confidence set containing about 60 percent of the mispredictions, then saturating counters appear to be adequate. However, the large maximum saturated counter region makes it impossible to partition with a larger low confidence set -- i.e. containing more than 60 percent or more of the mispredictions. Consequently, we might like to find a better method for cost-saving than saturating counters that maintains the zero bucket characteristics of the optimal curve.

Resetting Counters. A third algorithm was arrived at by examining the sorted lists of CIRs. In doing so, we observed that a few CIR patterns account for most of the dynamic branches. In particular those containing all zeros or a single one -- this reflects the high accuracy of the underlying branch predictor. This suggests that we can capture most of the information of full-length CIRs by only keeping track of the time the most recent misprediction has occurred, and this can be accomplished by incrementing a counter held in the CT each time the corresponding branch is predicted correctly, and resetting the counter to zero on any misprediction. The counter saturates at 16 -- to match the CIR and counter sizes used in the preceding subsections.

Fig. 8 shows the results with a resetting counter, as compared with the ideal reduction function, ones counting, and saturating counters. We see that the resetting counter works quite well. It tracks the ideal curve closely, and has the same zero bucket. In addition, as observed earlier for saturating counters, a resetting counters can replace the full-length CIRs in the CT, yielding a logarithmic cost savings. We conclude that resetting counters are probably the best choice for implementing confidence methods in a practical manner.

5.2. Threshold Granularity

For producing the confidence sets in practice, we note we are at the mercy of the natural "buckets" that result from the reduced CIRs. For example, if we were using resetting counters, we can only establish confidence sets at data points determined by the 17 resetting counter values. Table 1 contains the 17 data points defined by the best single level method using resetting counters that count from 0 to 16. These counters saturate at 16. The first column contains the counter value; the second is the misprediction rate when the counter has the shown in the first column. The third and fourth columns are the percentages of references and mispredictions that occur for the counter value, and the fifth and sixth columns are the cumulative reference and misprediction percentages from the top of the table. If we were to use a count value of 0 to define the low confidence set, then we could isolate 41.7 percent of the mispredictions to a set of 4.28 percent of the branch predictions. Of course, we would only need a single-bit "counter" to accomplish this. Similarly, if we were to use a count of 0 or 1, then we could isolate 57.9 percent of the mispredictions to within a set of 6.85 percent of the branch predictions, etc. The maximum count, 16, is equivalent to the zero bucket for full-length CIRs. Consequently, if we use counter values from 0 to 15, we can isolate 89.3 percent of the mispredictions to a set of 20.3 percent of the branches. In the relatively large zero bucket region defined by the saturated counter, we can not achieve any finer granularity. We could, however, use larger counters to get somewhat better granularity, but this approach is limited.

Table 1: Statistics for Resetting Counter Values

Count	Mis-pred. Rate	% Refs.	% Mis-preds.	Cum. % Refs.	Cum.% Mis-preds.
0	.376	4.28	41.7	4.28	41.7
1	.241	2.58	16.2	6.85	57.9
2	.174	1.91	8.64	8.76	66.5
3	.136	1.54	5.45	10.3	72.0
4	.112	1.31	3.80	11.6	75.8
5	.090	1.14	2.66	12.8	78.5
6	.079	1.03	2.12	13.8	80.6
7	.071	.936	1.73	14.7	82.3
8	.062	.860	1.39	15.6	83.7
9	.058	.798	1.19	16.4	84.9
10	.053	.744	1.02	17.1	85.9
11	.045	.699	.817	17.8	86.7
12	.043	.662	.746	18.5	87.5
13	.040	.628	.645	19.1	88.1
14	.039	.598	.599	19.7	88.7
15	.037	.571	.550	20.3	89.3
16	.005	79.7	10.7	100.	100.

Finally, note that we are using the same tables and reduction functions for all programs, yet we have been discussing confidence levels for a particular set of benchmarks. There can be variation in the confidence sets depending on the individual benchmark or program. Considering just the IBS benchmarks, Fig. 9 shows the confidence curves for the best (JPEG) and worst (GCC) performing IBS benchmarks using the best single level method with ideal reduction. We see that there is considerable variation. The zero buckets appear to contain approximately the same fractions of mispredictions, but the total number of branches in the zero bucket varies considerably. The importance of this characteristic will depend on the application to which the confidence

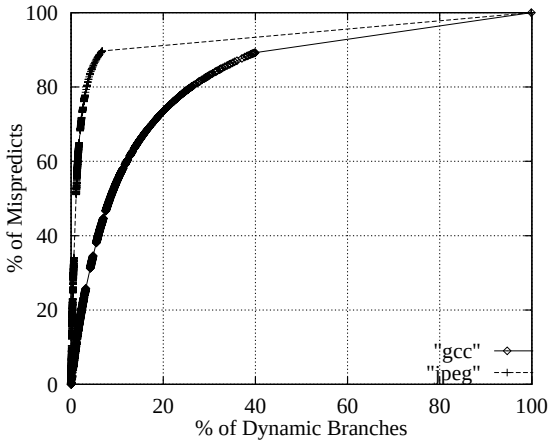

Fig. 9. Confidence graphs for best (JPEG) and worst (GCC) performing benchmarks. The best one level confidence method with ideal reduction is used.

mechanism is applied.

5.3. Cost Considerations

The study thus far has assumed fairly large (and relatively expensive) predictors and confidence methods. We did this to reduce aliasing effects so that we could get an idea of how a more pure implementation might perform. However, we should also consider smaller, less expensive implementations. We do not fully explore the design space here. Rather, we choose a particular small predictor and investigate the performance of the single level confidence method when aliasing is present. We use a series of small confidence history tables to illustrate the fall-off in performance as we reduce the table size.

For the smaller underlying branch predictor we use a gshare branch predictor with 4K entries consisting of saturating 2-bit counters. The predictor is accessed with the exclusive-OR of bits 13-2 of the branch program counter and 12 bits of global branch history. The misprediction rate of this predictor for the IBS benchmarks is 8.6 percent.

We implemented a confidence mechanism that uses the best single level method, which is accessed the same way as the gshare predictor. Resetting counters are held in the CT. We simulate tables that are the same size as the predictor, 4K entries, down to 128 entries. The results are plotted in Fig. 10.

We see that for the case where the confidence table is the same size as the predictor, relative performance is somewhat less than for the larger table studied. We note that the zero bucket regions -- i.e. where counters are at their maximum values -- contain about the same fraction of mispredictions as before, but there are fewer overall branches in this region. To put it another way, the low confidence sets contain about the same fraction of mispredictions, but they are larger sets. This occurs because of the aliasing that takes place. If any branch accessing the same table entry suffers a misprediction, then the counter resets, and it will take at least 16 correct predictions for the the counter to re-enter the saturated state. Hence, aliased counters are likely to spend more of their time in the non-saturated state. Nevertheless, the performance is still fairly good with 75 percent of the mispredictions being identified with 20 percent of the conditional branches. This suggests that the confidence method can be applied when smaller, less costly predictors are used. For the case where the prediction and confidence tables have the same number of entries, the cost of the confidence method is twice the underlying predictor (4-bit resetting counters versus 2-bit saturating counters).

From Fig. 10, we also see that performance diminishes in a well-behaved manner as the confidence table is reduced in size. Again, this loss is no doubt due to aliasing. We can not make any definite statement about any one table size being more cost-effective than another, however, because to do so would require some knowledge of the application where the confidence method is to be used. That is, a performance/simulation model of the application for which we are using the confidence method would have to be used to determine actual performance impact of the confidence sets.

5.4. The Effect of Initial State

In the course of this study, we found that the initial state of the CIRs held in the CT is important, because it

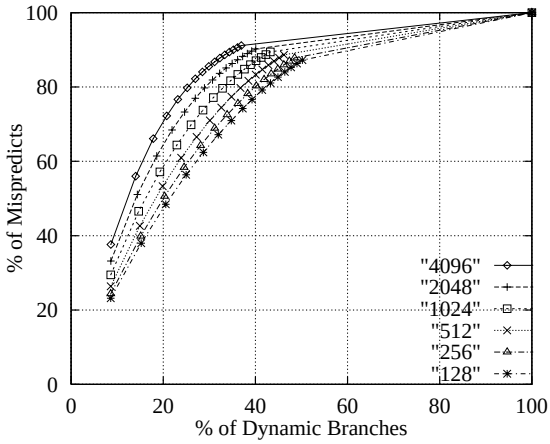

Fig. 10. Performance with small CIR tables; tables hold resetting counters, accessed with PC xor BHR.

takes a long time for the tables to build up history. There are a number of possibilities for initializing the CIRs. For example, one could initialize the CIRs to all 0s, all 1s, and random. Another alternative is to not initialize the CIRs between context switches, but we did not study this alternative.

Fig. 11 shows results for the ideal one level method using a CT with 2^{16} entries. We see that all 1's and random perform similarly, but all 0s does not perform nearly as well. As mentioned above, there are differences because the CT has a fairly "deep" memory, and initial state effects still appear even when the benchmarks are run to their full length. The all 0's initialization increases the number of branches in the zero bucket when a benchmark begins. At startup there is likely to be a higher number of mispredictions, but the initially zero CIRs will assign them a high confidence. On the other hand, a non-zero initial value will tend to assign a lower confidence. However, exactly which non-zero initial value is used does not seem to make much difference.

If better performance is provided by avoiding all zeros, then a reasonable alternative is to initialize only the "oldest" bit in the CIR to 1 and the others to zero. We refer to this as "lastbit", and the results are shown in Fig. 11 along with the other initial values. We see that the performance is essentially the same as for the other nonzero methods. From this experiment, we conjecture that one could probably leave the CIRs at their current values at the time of a context switch, except the oldest bit which should be initialized at 1. This would tend to simplify the initialization hardware and provide good performance.

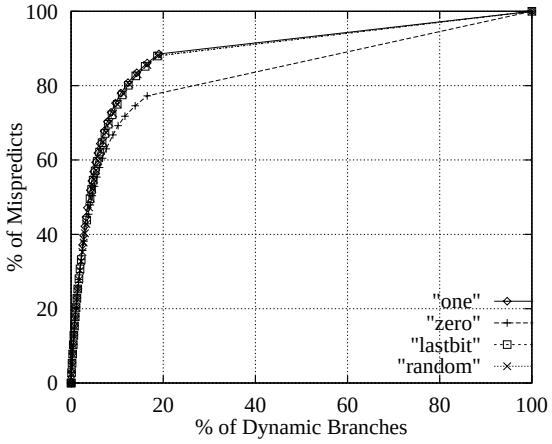

Fig. 11. The effect of different CT initializations; the best one level confidence method with ideal reduction is used.

6. Conclusions

For implementing confidence methods, there is a large design space -- probably as large as for branch prediction, and many branch prediction methods have been proposed and studied over the years. Consequently, the methods proposed here are by no means comprehensive. Our goal has been to introduce the concept of assigning confidence to branch predictions and to look at the characteristics of some methods. There are no doubt many other possible methods that can (and should) be explored.

Just looking at static branches indicates that some are predicted with higher confidence than others; but to better separate the low and high confidence predictions one must look at dynamic information. This is similar to the case with branch prediction where predictors taking the dynamic context into account perform better than static predictors.

Dynamic confidence methods are indeed better than static ones -- even the ideal static method. In general, it appears that the single level dynamic methods work as well as the two level methods and are less expensive. Of reduction functions, we found that resetting counters work well, and these counters can be embedded into the CIR table to reduce cost.

Most of our study was for large (64K) predictors and confidence tables. When using smaller (4K) predictors and smaller confidence tables (4K and smaller), we found that the confidence methods performed relatively worse -- the reason is that the use of resetting counters tends to amplify the negative effects of aliasing.

Of the applications we mentioned above, we are currently researching three out of the four. We are studying potential performance benefits of selective dual thread forking as well as aspects of the implementation. For this application, data for the single level methods indicate that if we fork a dual thread following 20 percent of the conditional branch predictions, we can capture over 80 percent of the mispredictions. We conjecture that this will be adequate to provide worthwhile performance gains.

We also have studies for the reverser and the hybrid prediction selector underway. The reverser application looks promising, but a key issue will be whether the cost/performance of a predictor plus reverser is better than the cost/performance of a more powerful predictor -- indeed, the predictor/reverser model could be considered a method of decomposing the design of a powerful predictor. The study of the hybrid prediction selector has just begun, but we are optimistic that work on branch confidence will lead to a systematic way of developing near-optimal selectors.

Acknowledgements

This work was supported in part by NSF Grant MIP-9505853 and by the U.S. Army Intelligence Center

and Fort Huachuca under Contract DABT63-95-C-0127 and ARPA order no. D346. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Army Intelligence Center and Fort Huachuca, or the U.S. Government. Eric Rotenberg is funded by an IBM graduate fellowship. Erik Jacobsen was funded by a Wisconsin/Hilldale undergraduate research scholarship

The authors would like to thank Joel Emer for helpful discussions while this work was in progress and for suggesting potential applications.

References

[1] M. Evers, P. Chang, and Y. Patt, ‘‘Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches,’’ *International Symposium on Computer Architecture*, pp. 3-11 , May 1996.

[2] Linley Gwennap, ‘‘MIPS R10000 Uses Decoupled Architecture,’’ *Microprocessor Report*, vol. 8, pp. 18-22, October 24, 1994.

[3] B. T. Hailpern and B. L. Hitson, ‘‘S-1 Architecture Manual,’’ CSL Report STAN-CS-79-715., Stanford University, January 1979.

[4] J. K. F. Lee and A. J. Smith, ‘‘Branch Prediction Strategies and Branch Target Buffer Design,’’ *Computer*, vol. 17, pp. 6 - 22, January 1984.

[5] S. McFarling, ‘‘Combining Branch Predictors,’’ Digital Western Research Lab Technical Note TN-36, June 1993.

[6] Motorola, ‘‘PowerPC 601 User’s Manual,’’ 1993, No. MPC601UM/AD.

[7] S.-T. Pan, K. So, and J. T. Rahmeh, ‘‘Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation,’’ *Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pp. 76-84, October 1992.

[8] Michael Slater, ‘‘AMD’s K5 Designed to Outrun Pentium,’’ *Microprocessor Report*, vol. 8, pp. 1,6-11, October 24, 1994.

[9] J. E. Smith, ‘‘A Study of Branch Prediction Strategies,’’ *Proc. Eighth Annual Symposium on Computer Architecture*, pp. 135-148, May 1981.

[10] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm, ‘‘Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor,’’ *Proc. 23rd Annual International Symposium on Computer Architecture*, pp. 191-202, May 1996.

[11] Richard Uhlig, David Nagle, Trevor Mudge, Stuart Sechrest, and Joel Emer, ‘‘Instruction Fetching: Coping with Code Bloat,’’ *Proc. 22nd Annual Symposium on Computer Architecture*, pp. 345-356, June 1995.

[12] A. K. Uht and V. Sindagi, ‘‘Disjoint Eager Execution: An Optimal Form of Speculative Execution,’’ *Proc. 28th Annual International Symposium on Microarchitecture*, pp. 313-325, November 1995.

[13] T. Y. Yeh and Y. N. Patt, ‘‘Two-Level Adaptive Branch Prediction,’’ *Proc. 24th Annual International Symposium on Microarchitecture*, November 1991.