Composite Confidence Estimators for Enhanced Speculation Control

Daniel A. Jiménez   Calvin Lin
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
{djimenez,lin}@cs.utexas.edu

Abstract

Many speculative microarchitectural techniques such as eager execution, value prediction, pipeline gating, and others rely on a confidence estimator to predict whether a speculative action will be beneficial. Since they cannot achieve perfect accuracy, confidence estimators must balance the cost of inhibiting a potentially useful speculation with the cost of a mis-speculation. The performance of confidence estimators with respect to these competing costs is quantified by two values: the SPEC*, i.e., the probability that an incorrect prediction is identified as low-confidence, and the* PVN*, i.e. the probability that a prediction identified as having low confidence will be incorrect. We propose a way to allow more effective use of speculation control techniques by combining multiple confidence estimators into a* composite confidence estimator. *This new class of confidence estimators provides increased performance and finer control over the trade-off between* SPEC *and* PVN.

The main contributions of this paper are twofold. First, we describe techniques for building efficient composite confidence estimators, and evaluate them with a previously proposed statistical methodology, emphasizing the relationship of SPEC *and* PVN*. Second, we use a detailed microarchitectural simulator to evaluate the ability of our estimator to support an energy reduction technique called pipeline gating. Using previous confidence estimators, pipeline gating reduces the amount of extra work due to mis-speculated instructions by 22%, with a reduction in IPC of 5%. With the same impact on IPC, our confidence estimators reduce extra work by 31%.*

1 Introduction

As high-performance microprocessors rely more and more on speculation to break control and data dependencies, confidence estimators will play a greater role in microarchitecture designs to control this speculation. Many microarchitectural techniques proposed recently depend on confidence estimation to control speculation. Some techniques use a confidence estimator to label a conditional branch prediction as having low or high confidence. For example, throttling instruction fetch when low-confidence branches are in the pipeline can save the energy wasted on mis-speculated instructions. Another example is eager execution, where instructions are fetched and executed down both paths when a low-confidence branch is encountered. A third example is a technique for boosting SMT performance by giving lower priority to threads executing several low-confidence unresolved branches.

For these techniques to be effective, the accuracy of the confidence estimator must be balanced between two measures. The first measure is the *predictive value of a negative estimate* (PVN), giving the probability that an estimate of low confidence indicates a misprediction. The second measure is the *specificity* (SPEC), giving the probability that a misprediction is estimated to have low confidence.

Unfortunately, PVN and SPEC are inversely proportional to one another, so we must rely on the flexibility and accuracy of the confidence estimator to find the right trade-off. Figure 1 shows the trade-off between SPEC and PVN for the *gshare* predictor using a confidence estimator we introduce. As the figure illustrates, we can have almost arbitrarily high PVN if we are willing to accept a very low SPEC, and vice-versa. The right trade-off for most applications is somewhere between these extremes. For instance, with the instruction fetch throttling example, when the PVN is too low, too few instructions are fetched and performance suffers. When the SPEC is too low, too many instructions are fetched and too much energy is wasted. In the eager execution example, if the

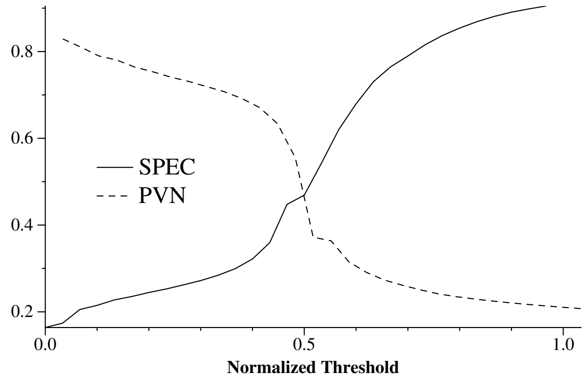

Figure 1. SPEC and PVN for *gshare* and Composite Estimator

SPEC is too low, then not enough opportunities for eager execution are found. If the PVN is too low, then the optimization is invoked on the wrong branches too frequently, wasting execution bandwidth that could have been devoted to single-threaded execution or other low-confidence branches.

The confidence estimation techniques proposed in existing research are too inflexible. They provide only coarse control over the balance between PVN and SPEC, and they have limited accuracy.

This paper shows that *composite confidence estimators*, which combine two or more confidence estimators, provide a finer degree of speculation control as well as increased levels of accuracy. We give experimental results showing the improvements of our estimators over previous work, and illustrate the improvements with a detailed cycle-level simulation of an energy reduction technique.

This paper makes the following contributions:

1. We describe techniques for building composite confidence estimators. These new confidence estimators are more accurate and flexible than previously proposed estimators, with little added complexity.

2. We show that the flexibility of a confidence estimator is an important aspect of its ability to control speculation because there is often a trade-off between competing penalties of overly optimistic and overly pessimistic confidence estimates. A confidence estimator with a wide range of PVN and SPEC values can be tuned to fit a particular application or set of applications.

3. We evaluate our new confidence estimation using a statistical methodology from previous work. We then improve on this methodology, showing that, for estimating confidence in branch predictions, the SPEC and PVN are the most important measures of confidence estimator performance.

4. We use a detailed microarchitecture simulator to evaluate the ability of our estimator to support an energy reduction technique called *pipeline gating*. Using previous confidence estimators [8] pipeline gating reduces the amount of extra work due to mis-speculated instructions by 22%, with a reduction in IPC of 5%. With the same impact on IPC, our confidence estimators reduce extra work by 31%.

2 Background and Related Work

In this section, we review several confidence estimation techniques that have been proposed previously, as well as several applications of confidence estimators. We also review a statistical framework, inspired by medical diagnostic tests, in which confidence estimators are evaluated.

2.1 Confidence Estimation

Confidence estimators provide a level of confidence in a prediction. In this paper, we focus on branch prediction, so our confidence estimators provide a level of confidence for whether or not a branch prediction is correct. We also focus on *dynamic confidence estimators*, that use run-time information to provide confidence estimates.

Confidence estimators are used to decide whether or not to take a particular action based on the confidence of a branch prediction. The confidence estimator produces a small integer value we call the *raw output* of the estimator. If this value is greater than a certain threshold, then the branch prediction is estimated to have high confidence. Figure 2 shows a block diagram of a dynamic confidence estimator. The structure is similar to a two-level adaptive branch predictor [20]. The branch history and branch PC are hashed to select an entry in a table of counters, whose exact behavior is a function of the particular confidence estimation scheme. The counter is taken as the raw output of the estimator and compared to a statically determined threshold, yielding an estimate of either high or low confidence. As branches are predicted, the branch predictor feeds information about its success or failure in predicting branches back to the confidence estimator. For example, a miss distance counter counts the number of branches correctly predicted since the last misprediction [9]. We choose a threshold value against which to compare the raw output. If the counter is higher than the threshold, then we estimate high confidence; otherwise we estimate low confidence. Dynamic confidence estimators have been suggested in recent research [9, 7, 12]. We describe several confidence estimators in Section 3.

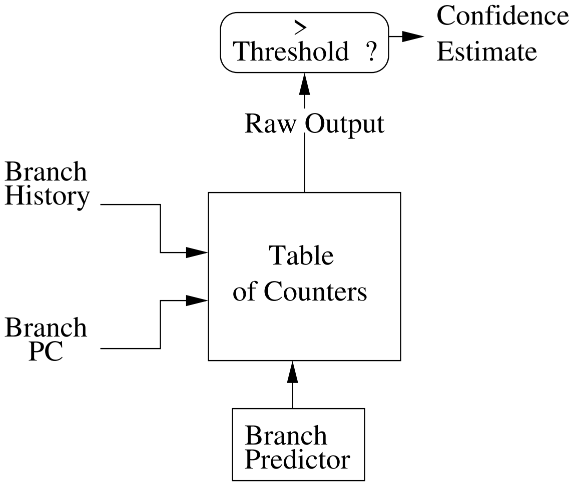

Figure 2. Dynamic Confidence Estimator Block Diagram

2.2 Evaluating Confidence Estimators

Manne *et al.* propose a statistical methodology for studying the performance of confidence estimators. We briefly review this methodology. In this framework, a confidence estimator returns one of two classifications: High Confidence (HC) or Low Confidence (LC). The branch prediction itself is labeled either Correct (C) or Incorrect (I), depending on the outcome of the branch. Four important statistics are associated with confidence estimators:

SENS. The sensitivity of a confidence estimator is defined as SENS $= P[HC|C]$, i.e., the probability that the confidence estimator reports a correctly predicted branch as having high confidence.

SPEC. The specificity is defined as SPEC $= P[LC|I]$, i.e., the probability that the confidence estimator reports an incorrectly predicted branch as having low confidence.

PVP. The predictive value of a positive estimate is defined as PVP $= P[C|HC]$, i.e., the probability that a prediction estimated to have high confidence is correct.

PVN. The predictive value of a negative estimate is defined as PVN $= P[I|LC]$, i.e., the probability that a prediction estimated to have low confidence is incorrect.

Dynamic confidence estimators based on comparing a raw output to a threshold can be tuned to yield different SENS, SPEC, PVP, or PVN values. For the applications mentioned above, having a high SPEC and PVN is important. For instance, for branch inversion, the PVN must be above $\frac{1}{2}$ or it would not make sense to invert predictions with low confidence since we would not be able to say that they are more likely wrong than not. For applications such as branch inversion and eager execution, there is one right value for the threshold: that which yields the highest performance.

For an application such as energy reduction, where some parts of the pipeline are throttled depending on the confidence values of branches in the pipeline, the issues are more subtle. In this case, we would like a confidence estimator capable of providing a wide range of PVN and SPEC values, since we want to find the right balance between saving energy and decreasing performance. When comparing dynamic confidence estimators as the threshold is varied, Manne *et al.* emphasize the relationship of PVP and PVN. However, since a high PVP is relatively easy to achieve and unimportant to several speculation techniques, we believe that emphasizing the relationship of PVN and SPEC is a better approach. Our results in Section 4 reflect this improved methodology.

2.3 Applications of Confidence Estimation

Researchers are increasingly relying on confidence estimation to boost performance and save energy in proposed future micro architectures. We review some of these applications.

Energy reduction. Grunwald and Manne introduced the technique of pipeline gating that uses a confidence estimator to reduce the energy wasted processing wrong-path instructions [8]. When there are more than a certain number of low-confidence branches in the pipeline, certain pipeline stages are "gated" or stalled, rather than wasting energy processing instructions that will be squashed when a misprediction is revealed. Baniasadi and Moshovos extend this work to consider other instruction flow information when deciding whether and how much to throttle instruction fetch [2]. For energy reduction, we need a confidence estimator with a high PVN to avoid an adverse impact on performance when too many branches are classified as low confidence. We also need a high SPEC so that enough opportunities for energy reduction can be identified.

Load value prediction. As with branch outcomes, load values have a great deal of regularity that can be exploited to improve performance [14]. Load value predictors can hide the latency of loads from memory. The decision of whether to predict a value or wait for the load to complete is made by a confidence estimator. Lipasti *et al.* suggest a simple confidence estimator that classifies loads as predictable, unpredictable, or almost predictable [14]. Burtscher and Zorn use profile based confidence estimators for load value prediction [5]. A mispredicted value has much the same effect as a mispredicted branch. Once the mis-speculated value can be compared with the actual value, all of the instructions that depended on the prediction have to be squashed and re-executed with the correct value. A high PVP and SENS makes sure that value prediction is applied when it is likely to be profitable. A high PVN suppresses value prediction when it suspects a misprediction, while a high SPEC makes sure that the decision to suppress value prediction was the right thing to do.

Eager execution. Branch mispredictions impose a steep penalty on performance. One way to avoid this penalty is to fetch and execute instructions from both directions of a branch until the branch is resolved. The processor executes several threads in parallel, spawning threads at branches and killing threads when the branches are resolved. This idea, in various forms, is known as eager execution [19, 13] and dual-path execution [6]. Since execution resources are limited, eager execution is restricted to branches with low confidence. If a low-confidence branch is fetched while the processor is already executing multiple threads, spawning yet another thread may not be feasible. Thus, a confidence estimator must be consulted to decide when to execute both paths leading from a branch. A high SPEC would enable eager execution for most of the mispredicted branches, while a high PVN would ensure that eager execution is exercised only when it is needed.

Boosting SMT Performance. Confidence estimators can be used to control the trade-off between speculation and simultaneous multithreading (SMT). Luo *et al.* propose a scheme for selecting instructions from various running threads based on the number of low-confidence unresolved branches they each have in the pipeline [15]. A thread with many low-confidence branches is given a lower priority, since instructions issued from that thread may be mis-speculations and will not contribute to the forward progress of the program. Once the low-confidence branches have resolved, the priority of the thread is increased. This scheme requires a high SPEC so that many opportunities for this optimization are exposed, and a high PVN to ensure that threads are only given a lower priority when they really are more likely to be mis-speculating.

Increasing Branch Predictor Accuracy. Under certain circumstances, a confidence estimator may indicate a high probability that a branch prediction is incorrect. If this probability is over 50%, it makes sense to invert the branch prediction. This technique is known as *branch inversion* [12] or *branch prediction reversal* [1]. For this technique to work, it is essential that the PVN be greater than 0.5. We also need the SPEC to be high so that enough incorrect predictions can be inverted to have a significant effect on performance.

3 Composite Confidence Estimators

In this section, we describe our technique for combining confidence estimators. We discuss our technique in an abstract sense, then describe several composite confidence estimators and branch predictors.

3.1 Combining Confidence Estimators

Confidence estimation is the task of classifying a branch as having either high or low confidence. Such a classifier produces a raw output that is roughly proportional to the probability that the branch is correct. A threshold is applied to this value to make the final classification. There are several techniques in the statistical and machine learning literature for combining classifiers for improved accuracy [18]. One of the simplest is to find the sum of the outputs of each classifier, then apply a threshold to that sum to make the classification. We use this technique for combining the outputs values of several confidence estimators. The resulting combination, along with an appropriately chosen threshold value, is a *composite confidence estimator*. Figure 3 shows the structure of a composite confidence estimator. Several confidence estimators are assembled into a single estimator by adding their respective raw outputs, which is then compared with a statically selected threshold.

3.2 Branch Predictors

Before describing the various confidence estimators, it is important to discuss the branch predictors for which we are assigning confidence. We choose three branch predictors from the literature for our evaluation of composite confidence estimators. Confidence estimation becomes harder as the branch predictor's accuracy improves [7].

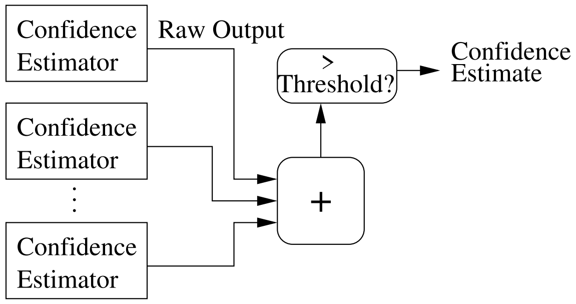

Figure 3. Composite Confidence Estimator Block Diagram

Thus, we choose a use a generous but realistic hardware budget to ensure that our results are conservative. Each of the predictors is allocated approximately four kilobytes of state, which is equivalent in size to the branch predictor in the Alpha 21264 [11], which, as of this writing, is the largest documented branch predictor in an existing microarchitecture.

Gshare. Based on the idea of two-level adaptive branch prediction [20], *gshare* indexes a pattern history table (PHT) of two-bit saturating counter with the exclusive-OR of a global history shift register and the branch program counter [16]. The high bit of the corresponding counter is taken as the prediction. A value of 1 means *predict taken*, while 0 means *predict not taken*. When a branch is executed, the history register and branch PC are again combined and used to index the PHT. The corresponding counter is incremented if the branch was taken, or decremented otherwise. The outcome of the branch is shifted into the history register, which records a 1 for *taken* and 0 for *not taken*. We model a *gshare* predictor with 16K entries.

Hybrid Predictor. Hybrid predictors combine two or more branch predictors to increase accuracy. We use a McFarling-style hybrid predictor [16] of the type implemented for the Alpha 21264 [11]. This predictor uses two branch prediction components: a 4K-entry GAg [21] predictor indexed by a global history shift register, and a 1K-entry PAg predictor, indexed by one of 1024 per-branch 10-bit history shift registers, combined with a 4K-entry chooser table. The PHT for the GAg predictor consists of two-bit saturating counters, while the PHT for the PAg component contains three-bit saturating counters.

Perceptron Predictor. As an alternative to branch predictors based on saturating counters, we evaluate composite confidence estimators with the *perceptron predictor*, a branch predictor based on neural learning [10]. The predictor uses the branch PC to index a table of perceptrons, which are vectors of small integer weights. The predictor computes the dot-product of the weights vector and a global branch history shift register, producing a signed integer value. If the value is at least 0, the branch is predicted to be taken, otherwise it is predicted not to be taken. Perceptron learning is used to update the weights vector when the magnitude of the dot-product value does not exceed a certain threshold, or when the prediction was incorrect. To update the perceptron, the elements of the weights vector are incremented or decremented depending on whether there was positive or negative correlation, respectively, between the corresponding bit in the history register and the branch outcome. One interesting aspect of this predictor is that the dot-product output is highly correlated with the probability that the branch is taken. Thus, this value has the potential to be used as the basis of a confidence estimator [10].

As branch predictors become more accurate, confidence estimation is harder because there are fewer mispredictions. Figure 4 shows the misprediction rates of the branch predictors simulated on the SPEC 2000 integer benchmarks, as well as the harmonic mean misprediction rate.

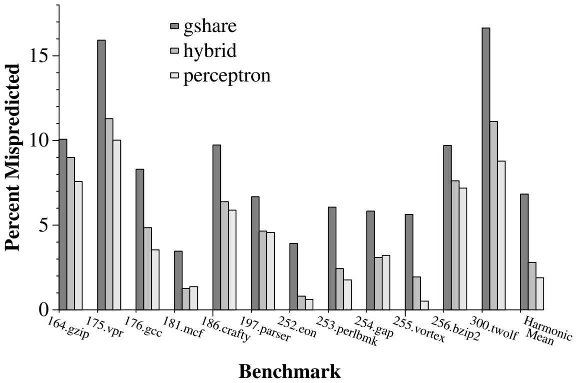

Figure 4. Misprediction Rates of Branch Predictors Simulated

3.3 Confidence Estimators

In this section, we describe several predictors from the literature that we use as the elements of our composite confidence estimators.

Enhanced JRS Estimator. Jacobsen *et al.* describe a confidence estimator based on counting the number of branch predictions made since a misprediction [9]. A table of miss distance counters (MDCs) is indexed by some combination of branch history and branch PC, much as in a two-level branch predictor. The raw output of the estimator is the MDC value from the table. The branch is labeled as high confidence if the raw output is above a statically determined threshold. Grunwald *et al.* call this confidence estimator a JRS estimator after the initials of the authors of the original paper describing its use and describe an enhanced version that updates the history register with the branch prediction in question before reading the MDC. This enhanced JRS estimator is shown to strictly outperform the earlier version [7], so we use it exclusively. We use four-bit counters for the MDC registers. We find no additional benefit from increasing this width.

Up/Down Counter Estimator. Klauser *et al.* introduce the use of *up/down* counters for confidence estimation [12]. This scheme is similar to the JRS estimator, but rather than resetting the counter on a misprediction, the counter is decremented. Thus, the counter records, for the short term, an approximation of the number of correct predictions for a particular combination of branch PC and history. Klauser *et al.* explore using only two-bit counters, but we have found additional benefit by using four bits.

Self-Estimator. Most branch prediction schemes already have a confidence estimator built-in for free: the values of the saturating counter used to make the prediction. In a situation where two-bit counters are used, we would expect a counter value of 3 to have higher confidence than a value of 2, since a value of 2 indicates more variability for the corresponding combination of branch PC and history. For a PHT-based scheme with n-bit saturating counters, if a branch is predicted based on the value c of a counter, we compute a value c' for the raw output such that:

$$c' = \begin{cases} c & \text{if the branch is predicted taken} \\ 2^n - c - 1 & \text{if the branch is predicted not taken} \end{cases}$$

We then estimate high confidence is c' is at least some threshold. For the McFarling hybrid predictor, we compute the sum of the corresponding c' values for the component GAg and PAg predictors, and apply a threshold.

Parameter	Configuration
L1 I-cache	64KB
L1 D-cache	64KB
L2 cache	1024KB
BTB	512 entry, 2-way set-assoc.
Issue width	8
Pipeline Depth	7

Table 1. Parameters Used for the Simulations

For the perceptron predictor, we use the magnitude of the dot-product value, scaled by simple shifting so that it is between 0 and 15, then apply a threshold. For PHT-based predictors, Grunwald *et al.* call this sort of confidence estimator a *saturating counters estimator*, and also explore a confidence estimator based on whether both or either component predictors of a hybrid predictor have high confidence [7].

4 Experimental Results

In this section, we evaluate several composite confidence estimators. We begin by reporting statistics on the performance of our new composite confidence estimators compared with previous work. This comparison takes place with respect to three branch predictors with increasing levels of accuracy. When then give results on an application of confidence estimation for an energy-saving technique, again showing results as we change the underlying branch predictor.

4.1 Methodology

We use the 12 SPEC 2000 integer benchmarks running under SimpleScalar/Alpha [4] to evaluate our confidence estimators. To better capture the steady-state performance behavior of the programs, our evaluation runs skip the first 500 million instructions, as several of the benchmarks have an initialization period (lasting fewer than 500 million instructions), during which branch prediction accuracy is unusually high. Each benchmark executes at least 300 million branches and over one billion instructions on the `ref` inputs before the simulation ends. Table 1 shows the microarchitectural parameters used for the simulations.

Branch history shift register length has been observed to have a significant impact on predictor accuracy [16], so for *gshare* we try all possible history lengths on the `train` inputs and keep the one with the lowest average misprediction accuracy. For the perceptron and McFarling predictors, we use configurations reported for the corresponding hardware budget in the literature [11, 10].

4.2 Confidence Estimators Simulated

We simulate the enhanced JRS (hereafter, simply JRS) and Up/Down confidence estimators, each using tables of 1024 4-bit counters and indexed using the method described in Section 3.3, consuming a small hardware budget of 512 bytes. We simulate the self-estimators of each branch predictor. We also simulate the following composite confidence estimators:

JRS + Up/Down. This estimator uses 512 4-bit miss distance counters and 512 4-bit Up/Down counters. Each table is indexed using the method described in Section 3.3. The raw output of the estimator is the sum of the indexed counters from each table.

JRS + Self. This estimator uses JRS estimator with 1024 counters. The raw output is the sum of the raw outputs of the JRS estimator and the self-estimator.

Up/Down + Self. This estimator uses an Up/Down estimator with 1024 counters. The raw output is the sum of the raw outputs of the Up/Down estimator and the self-estimator.

JRS + Up/Down + Self. This estimator adds the raw output of the JRS + Up/Down estimator to the raw output of the self-estimator.

4.3 Statistical Results

We report statistics for the entire range of threshold values for each confidence estimator and branch predictor. We examine plots of these statistics using techniques from previous work, then look at improved plots that yield more information.

4.3.1 PVP vs. PVN

We begin with the same statistical evaluation given in other work [7]. Without having a particular application in mind, we can consider one confidence estimator to be better than another if it has higher PVN and PVP values. Figure 5 shows a graph with PVP plotted against PVN for several of the confidence estimators. From this graph, we see that the individual JRS and Up/Down estimators have high PVP and PVN values compared with the composite Up/Down + JRS estimator, but the composite estimator has a wider range of PVP and PVN values, making it more flexible.

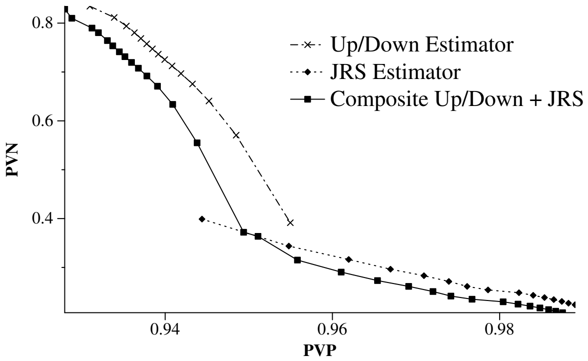

Figure 5. PVP vs. PVN for *gshare*

4.3.2 Distribution of Confidence Estimates

The performance of a confidence estimator cannot be summarized with a single type of statistic. For instance, for many optimizations it is important for the confidence estimator to have a high PVN. However, it is meaningless to say that a confidence estimator has a high PVN and high PVP without also discussing the SPEC value. The predictive value of a negative (i.e. low-confidence) estimate can be made almost arbitrarily high if we allow many false positives, i.e., if the SPEC is low. Moreover, since branch predictors generally have high accuracy, it is easy to achieve a high PVP. Note the small range of PVP values in Figure 5.

To illustrate the nature of this problem, Figure 6 shows a histogram of the cumulative percentage of *gshare*-predicted branches estimated to have low confidence for varying thresholds. For the each estimator, as the threshold is increased, more branches are estimated to have low confidence. The JRS estimator overestimates the number of mispredicted branches, consistently labeling many more branches as having low confidence for each threshold value. The Up/Down estimator underestimates mispredictions, labeling many fewer branches as having low confidence. The composite JRS + Up/Down estimator strikes a balance between the two. From this histogram we cannot directly infer that the composite estimator is better than the other two, but we see the potential for a more even-handed distribution of confidence estimates.

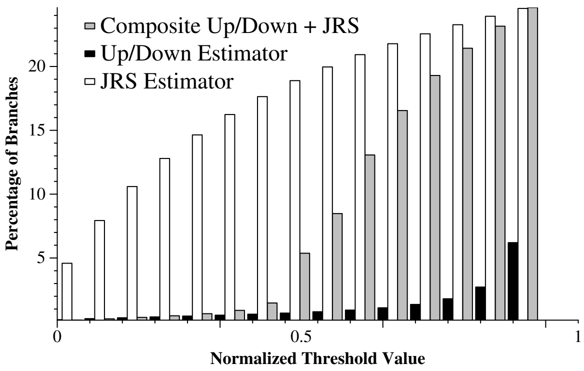

Figure 6. Distribution of Confidence Estimates

4.3.3 PVN vs. SPEC

To get a more informative comparison of confidence estimators, we must compare PVN with SPEC. Both of these values are important for many applications that use confidence estimation when deciding whether to take an action, such as pipeline gating or eager execution. We need a high PVN so that we do not needlessly take the action, and we need a high SPEC so that we have ample opportunity to take the action when it is appropriate.

Figure 7 shows a plot of the SPEC values of several confidence estimators for *gshare* against their respective PVN values for the entire range of feasible thresholds. Higher values in both the x- and y-axes are better. From the graph, we can see that both the JRS and Up/Down estimators are better than the composite, but only in certain narrow and mutually exclusive ranges. The composite JRS + Up/Down estimator has slightly lower PVN and SPEC, but covers a much wider range of values. Thus, the composite estimator is likely to be more appropriate for an application that requires flexibility in the confidence estimator. In Section 5, we give an example of such an application.

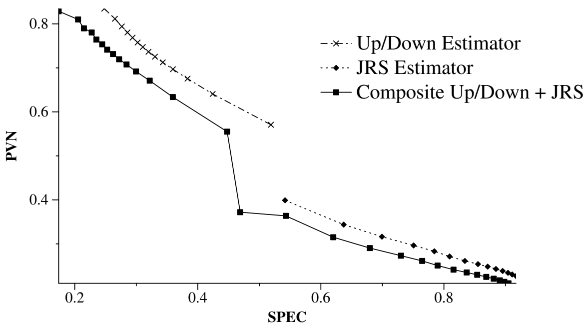

Figure 7. SPEC vs. PVN for *gshare*

4.3.4 Other Branch Predictors

Thus far, we have only applied composite confidence estimators to the *gshare* branch predictor. However, many other branch predictors with better accuracies have been proposed and implemented. We evaluate our confidence estimators with the McFarling hybrid predictor and the perceptron predictor. As we observed in Section 4.2, both of these predictors have robust self-estimators, i.e., the predictor's internal state can produce a raw output capable of generating a confidence estimate.

Figure 8 shows a graph of SPEC vs. PVN for the perceptron predictor. As we observed previously for *gshare*, the JRS and Up/Down estimators separately have higher SPEC and PVN than the composite JRS + Up/Down estimator in specific areas. However, when we add the self-estimator into the raw output, the composite JRS + Up/Down + Self estimator has higher SPEC and PVN than any of the other estimators at all threshold values.

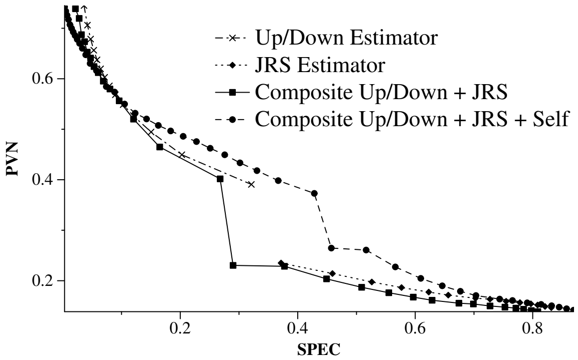

Figure 8. SPEC vs. PVN for a Perceptron Predictor

Figure 9 shows a graph of SPEC vs. PVN for the McFarling-style hybrid predictor. At some points, the combined JRS + Up/Down + Self estimator is more accurate than the other estimators. Again, both composite estimators have wider ranges than the individual estimators.

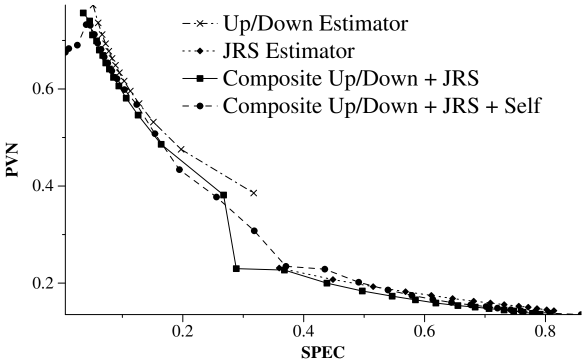

Figure 9. SPEC vs. PVN for a McFarling Hybrid Predictor

5 Application of Composite Confidence Estimators

Although we can use statistical measures such as SPEC and PVN to evaluate new confidence estimators, the best way to compare confidence estimators is to use them in an application. In this section, we give results of a detailed cycle-level simulation of an energy reduction optimization using composite confidence estimators.

5.1 Pipeline Gating for Energy Reduction

Manne *et al.* propose a technique called pipeline gating for reducing the energy demands of high performance processors without significantly reducing performance [8]. The idea is to control rampant speculation by using a confidence estimator to throttle various stages of the pipeline when several unresolved branches with low confidence are in-flight. When a branch misprediction seems imminent, it does not make sense to waste energy by continuing to fetch and execute instructions whose results are likely to be thrown away. Other research has proposed similar energy reduction techniques [2], and a similar mechanism is used in G3 and G4 PowerPC processors [17] to trigger instruction fetch throttling when temperature exceeds a certain threshold.

We simulate a form of pipeline gating using our confidence estimators. We modify SimpleScalar/Alpha to cease instruction fetch when there are three or more unresolved branches with low confidence. Instruction fetch continues when enough branches have resolved so that there are fewer than three unresolved branches with low confidence. Manne *et al.* find that gating with three low-confidence branches yields the best energy reduction. Having tried other values, we reach the same conclusion. We simulate pipeline gating with all threshold values for each confidence estimator. Note that there is no "best" threshold value. Since the threshold controls the trade-off between energy and performance, the choice of threshold should be made to fit the particular application.

5.1.1 Reduction in Extra Work

The goal of pipeline gating is to eliminate as much needless work as possible. We measure this extra work as the number of useless instructions per cycle, i.e., the average number of all executed instructions minus the number of committed (i.e., useful) instructions per cycle. Figure 10 shows a graph of the decrease in performance incurred by pipeline gating plotted against the decrease in the amount of harmonic mean extra work when the branch predictor is *gshare*. Each point on the curves in the graph represents a different threshold value for the confidence estimator. Depending on the energy constraints, a microarchitect may choose to set the threshold low, for higher performance, or high, for higher energy savings. The graph shows the performance of two confidence estimators: the JRS estimator used in the original pipeline gating work [8] and a composite estimator combining JRS and the Up/Down estimator. With a decrease in IPC of 2.4%, the JRS estimator is able to eliminate 11.7% of the extra instructions. With the same decrease in IPC, the composite estimator eliminates 18.7% of the extra instructions. At each point on the graph, the composite estimator provides greater energy savings for the same impact on performance. Note that, although we tried every possible threshold value for each confidence estimator, the graphs only show those for which the percentage decrease in IPC is in a narrow range that we believe is acceptable. Notice also that the amount of extra work decreases almost monotonically with the decrease in IPC, although for some threshold values there is a slight relative increase in IPC over the previous threshold value. This is because pipeline gating sometimes actually helps performance by relieving contention for execution resources caused by mis-speculated instructions.

Figure 11 compares the performance of the same two confidence estimators when *gshare* is replaced by a McFarling hybrid branch predictor. This graph also shows the performance of a third composite confidence estimator that uses the internal state of the branch predictor as well as an external confidence estimator. We learn two important facts from this graph. First, both composite estimators achieve greater energy savings for the same reduction in IPC than baseline JRS estimator. Second, by adding the self-estimator for the hybrid predictor, we increase the range over which we can trade off changes between energy and performance. For instance, with a 0.5% decrease in IPC, the JRS + Up/Down + Self estimator yields a 7.1% decrease in the amount of extra work performed. This level of fine-tuning is simply not available with the other estimators. With the lowest possible thresholds, we must still sacrifice 1% of IPC to achieve any energy savings.

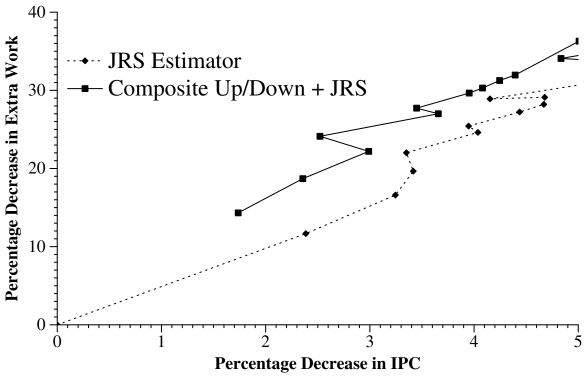

Figure 10. Decrease in Performance vs. Decrease in Extra Work for *gshare*

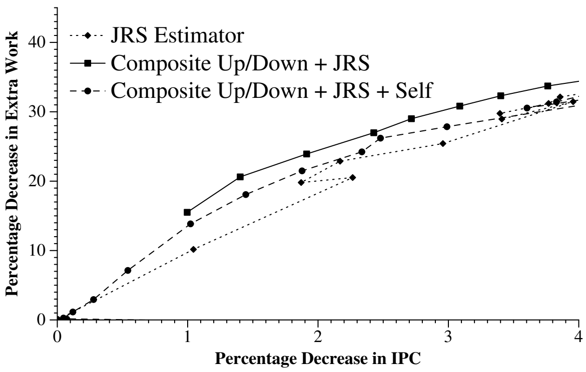

Figure 11. Decrease in Performance vs. Decrease in Extra Work for Hybrid Predictor

Figure 12 shows a plot of the decrease in IPC against the decrease in extra work for the perceptron predictor. The perceptron predictor is the most accurate of the three branch predictors simulated, and thus presents the most difficult situation from which to extract energy savings from avoiding useless work. Still, composite confidence estimators are able to provide a wide range of IPC vs. energy savings. The lowest threshold JRS estimator yields a decrease of 13.1% in extra work, at a cost of a 3.4% lower IPC. The composite JRS + Up/Down + Self estimator, now using the scaled perceptron output as a component, achieves a greater savings of 16.5%, with a smaller performance penalty of only 2.6%. Furthermore, the JRS + Up/Down + Self estimator provides a much wider range of energy savings than either the JRS or JRS + Up/Down estimators, allowing more fine-tuning of the pipeline gating technique. Note that the perceptron self-estimator provides a modest savings in energy without the extra hardware of a composite estimator.

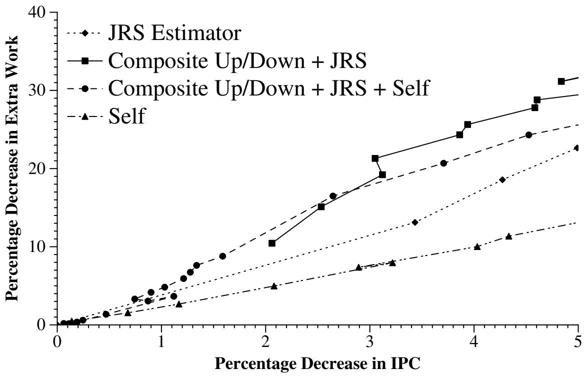

Figure 12. Decrease in Performance vs. Decrease in Extra Work for Perceptron Predictor

The potential for energy reduction is due to the number of mis-speculated instructions executed per cycle. Figure 13 shows the number of mis-speculated instructions per cycle for each benchmark using the perceptron predictor. The base case of no pipeline gating is shown, as well as the results for three confidence estimators that each reduce IPC by at most 5%. For `197.parser`, 2.0 instructions are wasted on each cycle in the base case. With the JRS estimator, only 1.19 extra instructions are wasted per cycle, a reduction of 40% over the base case. The composite JRS + Up/Down estimator reduces the number of mis-speculated instructions by 50% to 1.0 per cycle.

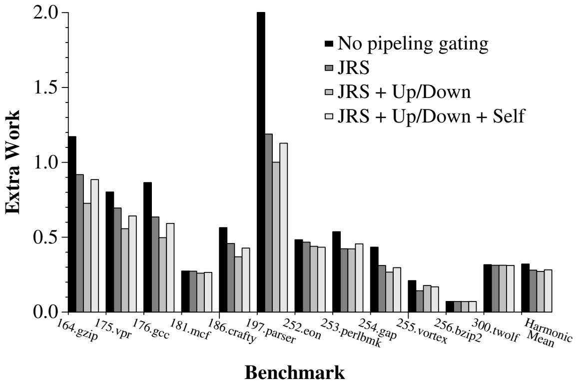

Figure 13. Extra Instructions per Cycle, Perceptron Predictor

5.2 Implementation

One concern when considering a new hardware mechanism is the cost in terms of transistors and power. The additional cost of our new confidence estimators is minimal. For each confidence estimator that we have studied in this paper, the hardware budget does not exceed 512 bytes of SRAM. Since we suggest that our designs can be used with an energy saving technique, it is important to note that the additional hardware itself will contribute a small amount to the energy requirements of the processor. To provide perspective, we used the Wattch microarchitecture simulator to gather statistics on power [3]. Using this tool, we find that a hybrid branch direction predictor (i.e., not including the BTB) with twice the hardware budget of our confidence estimators consumes a negligible 0.32% of the total power of the simulated microprocessor. The most complex of our designs adds two 5-bit adders to this budget.

6 Conclusions

Confidence estimation is a microarchitectural technique that enhances speculation by predicting whether the speculation will be useful. Composite confidence estimators exploit the best characteristics of multiple confidence estimators to provide enhanced control over speculation. Composite confidence estimators are able to achieve high degrees of accuracy even when misprediction rates are low, unlike previously proposed estimators. We have shown that our new estimators are able to give a wider range of control over the trade-off between SPEC and PVN as well as increased accuracy in both dimensions. Using a cycle-level microarchitectural simulator, we have shown how our new estimators enable pipeline gating to deliver more levels of energy savings with less sacrifice in performance. The implications of this and other work in confidence estimation reach far beyond a single application. For future work, we plan on more fully exploring the space of composite confidence estimators as well as measuring the performance of confidence estimators with respect to a wide variety of microarchitectural applications.

References

[1] Juan L. Aragón, José González, José M. Garcia, and Antonio González. Confidence estimation for branch prediction reversal. In *Proceedings of the 8th International Conference on High Performance Computing*, December 2001.

[2] Amirali Baniasadi and Andreas Moshovos. Instruction flow-based front-end throttling for power-aware high-performance processors. In *International Symposium on Low Power Electronics and Design (ISPLED)*, August 2001.

[3] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture*, Vancouver, British Columbia, June 2000.

[4] Doug Burger and Todd M. Austin. The simplescalar tool set version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin, June 1997.

[5] Martin Burtscher and Benjamin G. Zorn. Prediction outcome history-based confidence estimation for load value prediction. *Journal of Instruction-Level Parallelism*, 1, May 1999.

[6] Matthew Farrens, Timothy Heil, James E. Smith, and Gary Tyson. Restricted dual path execution. Technical Report CSE-97-18, Computer Science Department, University of California, Davis, November 1997.

[7] Dirk Grunwald, Artur Klauser, Srilatha Manne, and Andrew Pleszkun. Confidence estimation for speculation control. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 122–131, June 27–July 1 1998.

[8] Dirk Grunwald and Srilatha Manne. Pipeline gating: Speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 27–July 1 1998.

[9] Erik Jacobsen, Eric Rotenberg, and James E. Smith. Assigning confidence to conditional branch predictions. In *Proceedings of the 29th Annual International Symposium on Microarchitecture*, December 1996.

[10] Daniel A. Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture*, January 2001.

[11] Richard E. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[12] Artur Klauser, Srilatha Manne, and Dirk Grunwald. Selective branch inversion: Confidence estimation for branch predictors. *International Journal of Parallel Programming*, 29(1):81–110, February 2001.

[13] Artur Klauser, Abhijit Paithankar, and Dirk Grunwald. Selective eager execution on the polypath architecture. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.

[14] Mikko H. Lipasti, Christopher B. Wilderson, and John Paul Shen. Value locality and load value prediction. In *Proceedings of the 7th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, October 1996.

[15] Kun Luo, Manoj Franklin, Shubhendu S. Mukherjee, and Andre Seznec. Boosting smt performance by speculation control. In *Proceedings of the International Parallel and Distributed Processing Symposium*, April 2001.

[16] Scott McFarling. Combining branch predictors. Technical Report TN-36m, Digital Western Research Laboratory, June 1993.

[17] Hector Sanchez, Belli Kuttanna, Tim Olson, Mike Alexander, Gian Gerosa, Ross Philip, and Jose Alvarez. Thermal management system for high performance PowerPC microprocessors. In *Proceedings of COMPCON '97*, February 1997.

[18] Kagan Tumer and Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science, Special issue on combining artificial neural networks: ensemble approaches*, 8(3,4), December 1996.

[19] Augustus K. Uht and Vijay Sindagi. Disjoint eager execution: An optimal form of speculative execution. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, December 1995.

[20] T.-Y. Yeh and Yale N. Patt. Two-level adaptive branch prediction. In *Proceedings of the 24th ACM/IEEE Int'l Symposium on Microarchitecture*, November 1991.

[21] T.-Y. Yeh and Yale N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, May 1993.