# Composite Confidence Estimators for Enhanced Speculation Control

Daniel A. Jiménez
Department of Computer Science
The University of Texas at San Antonio
San Antonio, Texas, U.S.A
djimenez@acm.org

***Abstract*—This paper proposes 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 improved performance and finer speculation control.**

**This paper makes three contributions. First, we describe techniques for building efficient composite confidence estimators. Second, we present an improved statistical methodology for evaluating confidence estimators. Finally, 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%.**

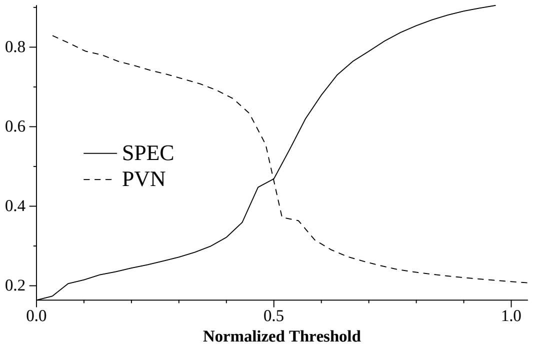

Figure 1. SPEC and PVN for *gshare* and Composite Estimator

## I. Introduction

Confidence estimation is a microarchitectural technique that allows control over speculation by predicting whether the speculation will be useful. Many proposed microarchitectural techniques 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.

### *A.* PVN *vs.* SPEC

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.

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 a high PVN if we can accept a very low SPEC, and vice-versa. The right trade-off for most applications is somewhere in 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.

### *B. New Confidence Estimators*

Previously proposed confidence estimation techniques provide only coarse control between PVN and SPEC and have limited accuracy. We propose combining two or more confidence estimators into a *composite confidence estimator*. With the same hardware budget and minimal extra complexity, composite confidence estimators provide a finer degree of speculation control, as well as increased accuracy in the confidence estimates. Our experimental results show the improvements of our estimators over previous work. We 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 present an improved statistical methodology for evaluating confidence estimators. We show that this methodology is superior to previous approaches because it emphasizes the relationship between SPEC and PVN. 3) 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 [1] 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%.

## II. 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 in which confidence estimators are evaluated.

### A. Confidence Estimation

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

The confidence estimator produces a small integer *raw output*. 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 (i.e. run-time) confidence estimator. The structure is similar to a two-level adaptive branch predictor [2]. The branch history and branch PC are hashed to select an entry in a table of counters whose behavior is a function of the particular confidence estimation scheme. The raw output of the estimator is 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 [3]. Dynamic confidence estimators have been suggested in previous work [3], [4], [5], [6]. We describe several confidence estimators in Section III.

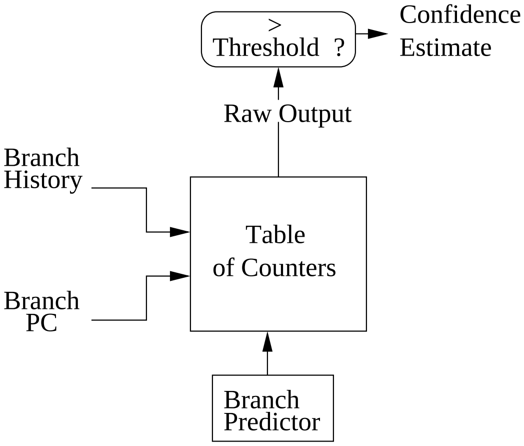

Figure 2. Dynamic Confidence Estimator Block Diagram

### B. Evaluating Confidence Estimators

Manne *et al.* propose a statistical methodology for studying the performance of confidence estimators. 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 ($I$). 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 a correctly predicted branch is predicted to have high confidencee

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 threshold comparison can be tuned to yield different SENS, SPEC, PVP, or PVN values.

For an application such as energy reduction, where parts of the pipeline are throttled depending on the confidence values of branches in the pipeline, 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 IV reflect this improved methodology.

### C. Applications of Confidence Estimation

Recent microarchitecture research has made use of confidence estimation in many ways, such as energy reduction [1], [7], [6], load value prediction [8], [9], [10], eager/polypath execution [11], [12], [13], increasing branch predictor accuracy [5], and more. We review some of these applications.

*Energy reduction.:* Grunwald and Manne introduced *pipeline gating* using a confidence estimator to reduce the energy wasted processing wrong-path instructions [1]. When there are at least a certain number of low-confidence branches in the pipeline, some pipeline stages are gated (i.e. stalled), rather than wasting energy processing mis-speculated instructions. Baniasadi and Moshovos extend this work to consider other instruction flow information when deciding whether and how much to throttle instruction fetch [7]. 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 to identify enough opportunities for energy reduction.

*Load value prediction.:* Load values have a great deal of regularity that can be exploited to improve performance [8]. 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 confidence estimator that classifies loads as predictable, unpredictable, or almost predictable [8]. Burtscher and Zorn use profile based confidence estimators for load value prediction [9]. 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 [11], [12] and dual-path execution [13]. 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. A high SPEC enables eager execution for most of the mispredicted branches, while a high PVN ensures that eager execution is exercised only when it is needed.

*Increasing Branch Predictor Accuracy.:* A confidence estimator might 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* [5]. This technique requires a PVN greater than 0.5 and a high SPEC so that enough incorrect predictions can be inverted to have a significant effect on performance.

## III. 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.

### A. 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 [14]. One of the simplest is to take 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.

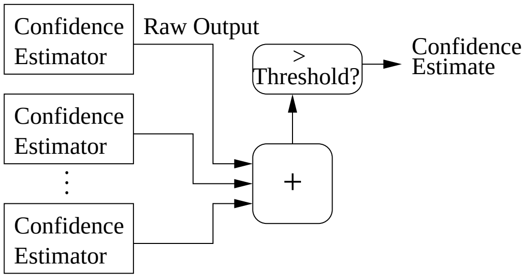

Figure 3. Composite Confidence Estimator Block Diagram

### B. 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 [4]. Thus, we choose a use a 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 [15].

*Gshare.:* Based on the idea of two-level adaptive branch prediction [2], *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 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. This predictor uses two branch prediction components: a 4K-entry GAg [17] predictor indexed soley by the history register, and a 1K-entry PAg predictor, indexed by one of 1024 per-branch 10-bit history 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 [18]. 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 [18].

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 arithmetic mean misprediction rate.

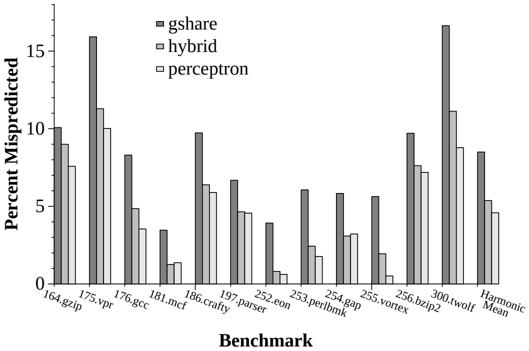

Figure 4. Misprediction Rates of Branch Predictors Simulated

## C. 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 [3]. A table of miss distance counters (MDCs) is indexed by combining branch history with branch PC. The output of the estimator is the MDC value from the table is above a statically determined threshold. Grunwald *et al.* call this the JRS confidence estimator after the initials of the original authors, and describe an enhanced version that updates the history register with the branch prediction in question before reading the MDC; we study this enhanced version with four-bit counters.

*Up/Down Counter Estimator.:* Klauser *et al.* introduce *up/down* counters for confidence estimation [5]. This scheme is similar to the JRS estimator, but the counter is decremented instead of cleared on a misprediction. Klauser *et al.* explore using only two-bit counters, but we have found additional benefit by using four bits.

*Self-Estimator.:* 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 if $c'$ exceeds some threshold. For the McFarling hybrid predictor, we compute the sum of the corresponding $c'$ values for the component predictors and apply a threshold. For the perceptron predictor, we use the magnitude of the dot-product value, scaled by shifting to 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* [4].

## IV. Experimental Results

In this section, we evaluate several composite confidence estimators. We report statistics on the performance of the confidence estimators.

### A. Methodology

We use the 12 SPEC 2000 integer benchmarks running under SimpleScalar/Alpha [19] to evaluate our confidence estimators. 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 I 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

| 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 I
PARAMETERS USED FOR THE SIMULATIONS

*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 [15], [18].

### *B. 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 III-C, 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 III-C. 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.

### *C. 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.

*1)* PVP *vs.* PVN*:* We begin with the same statistical evaluation given in other work [4]. 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.

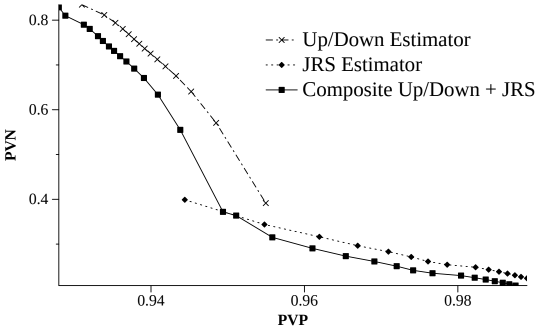

Figure 5. PVP vs. PVN for *gshare*

*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.

*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

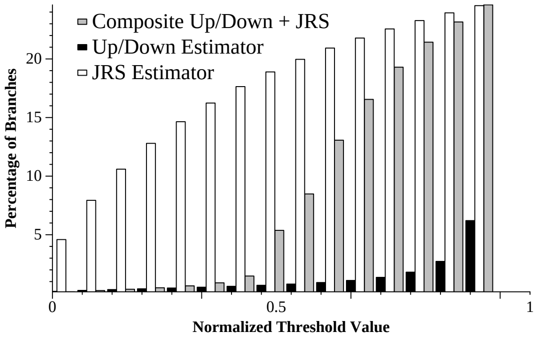

Figure 6. Distribution of Confidence Estimates

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 V, we give an example of such an application.

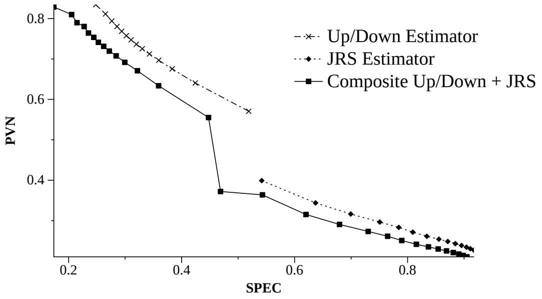

Figure 7. SPEC vs. PVN for *gshare*

*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 IV-B, 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.

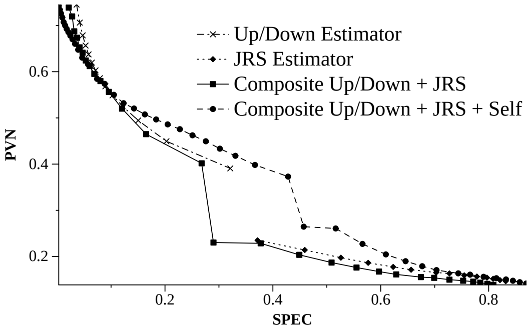

Figure 8. SPEC vs. PVN for a Perceptron Predictor

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.

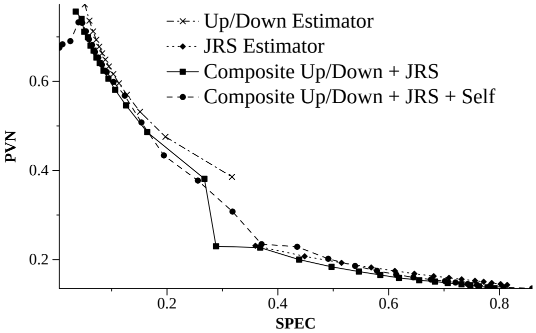

Figure 9. SPEC vs. PVN for a McFarling Hybrid Predictor

## V. Application of Composite Confidence Estimators

Although we can compare confidence estimators with one another to get an idea of which one is better, it is difficult to tell how much better without actually using the estimators in an application. In this section, we give results of a detailed cycle-level simulation of an energy reduction optimization using composite confidence estimators.

### A. 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 [1]. 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 [7], and a similar mechanism is used in G3 and G4 PowerPC processors [20] to trigger instruction fetch throttling when temperature exceeds a certain threshold. Another contribution to this line of research comes from Seznec and Vandierendonck who propose controlling the instruction fetch rate using confidence estimation to reduce wrong-path speculation and save energy [6]. Branchtap [21] is another technique that throttles speculation based on confidence to reduce the cost of recovering from mispredictions[1].

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.

*1) Reduction in Extra Work:* The goal of pipeline gating is to eliminate as much needless work as possible. We measure this extra 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 plot of the decrease in IPC against the decrease in extra work for the perceptron predictor[2]. 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.

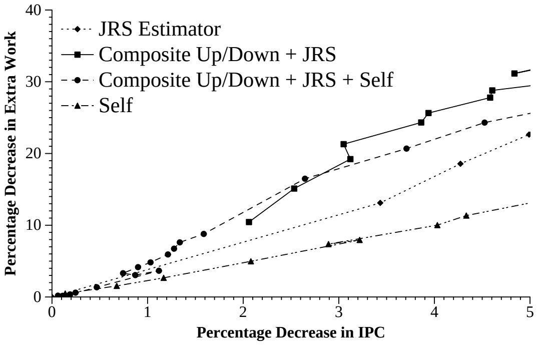

Figure 10. Decrease in Performance vs. Decrease in Extra Work for Perceptron Predictor

The potential for energy reduction is due to the number of mis-speculated instructions executed per cycle. Figure 11 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.

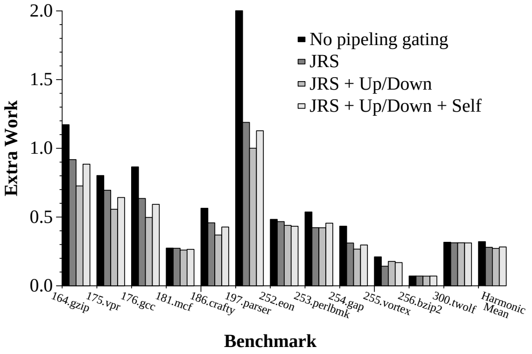

Figure 11. Extra Instructions per Cycle, Perceptron Predictor

## B. 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 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

[1] The work by Seznec and Vandierendonck as well as the work by Akl and Moshovos both cite the technical report version of this paper.

[2] For space reasons, we omit discussion of decrease in extra work for *gshare* and hybrid predictors.

to the energy requirements of the processor. To provide perspective, we used the Wattch microarchitecture simulator to gather statistics on power [22]. 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.

## VI. Conclusion

As microprocessors rely more on speculation to break control and data dependencies, confidence estimators will play a greater role in microarchitecture designs. Composite confidence estimators exploit the best characteristics of multiple 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.

## VII. Acknowledgments

This research was supported by the following grants from the National Science Foundation: CCR-0311091, CCF-0545898/0931874, CRI-0751138. Thanks to Calvin Lin helpful feedback on an early draft of this paper.

## References

[1] D. Grunwald and S. Manne, “Pipeline gating: Speculation control for energy reduction,” in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 27–July 1 1998.

[2] T.-Y. Yeh and Y. N. Patt, “Two-level adaptive branch prediction,” in *Proceedings of the 24th ACM/IEEE International Symposium on Microarchitecture*, November 1991, pp. 51–61.

[3] E. Jacobsen, E. Rotenberg, and J. E. Smith, “Assigning confidence to conditional branch predictions,” in *Proceedings of the 29th Annual International Symposium on Microarchitecture*, December 1996, pp. 142–152.

[4] D. Grunwald, A. Klauser, S. Manne, and A. Pleszkun, “Confidence estimation for speculation control,” in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 27–July 1 1998, pp. 122–131.

[5] A. Klauser, S. Manne, and D. Grunwald, “Selective branch inversion: Confidence estimation for branch predictors,” *International Journal of Parallel Programming*, vol. 29, no. 1, pp. 81–110, February 2001.

[6] H. Vandierendonck and A. Seznec, “Fetch gating control through speculative instruction window weighting,” in *2007 International Conference on High Performance Embedded Architectures and Compilers (HiPEAC 2007)*, 2007, pp. 120–135.

[7] A. Baniasadi and A. 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.

[8] M. H. Lipasti, C. B. Wilderson, and J. P. 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.

[9] M. Burtscher and B. G. Zorn, “Prediction outcome history-based confidence estimation for load value prediction,” *Journal of Instruction-Level Parallelism*, vol. 1, May 1999.

[10] M. Black and M. Franklin, “Neural confidence estimation for more accurate value prediction,” *Lecture Notes in Computer Science: High Performance Computing – HiPC 2005*, vol. 3769/2005, pp. 376–385, 2006.

[11] A. K. Uht and V. Sindagi, “Disjoint eager execution: An optimal form of speculative execution,” in *Proceedings of the 28th Annual International Symposium on Microarchitecture*, December 1995.

[12] A. Klauser, A. Paithankar, and D. Grunwald, “Selective eager execution on the polypath architecture,” in *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.

[13] M. Farrens, T. Heil, J. E. Smith, and G. Tyson, “Restricted dual path execution,” Computer Science Department, University of California, Davis, Tech. Rep. CSE-97-18, November 1997.

[14] K. Tumer and J. Ghosh, “Error correlation and error reduction in ensemble classifiers,” *Connection Science, Special issue on combining artificial neural networks: ensemble approaches*, vol. 8, no. 3,4, December 1996.

[15] R. E. Kessler, “The Alpha 21264 microprocessor,” *IEEE Micro*, vol. 19, no. 2, pp. 24–36, March/April 1999.

[16] S. McFarling, “Combining branch predictors,” Digital Western Research Laboratory, Tech. Rep. TN-36m, June 1993.

[17] T.-Y. Yeh and Y. 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.

[18] D. A. Jiménez and C. Lin, “Dynamic branch prediction with perceptrons,” in *Proceedings of the 7th International Symposium on High Performance Computer Architecture (HPCA-7)*, January 2001, pp. 197–206.

[19] D. Burger and T. M. Austin, “The SimpleScalar tool set version 2.0,” Computer Sciences Department, University of Wisconsin, Tech. Rep. 1342, June 1997.

[20] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez, “Thermal management system for high performance PowerPC microprocessors,” in *Proceedings of COMPCON ’97*, February 1997.

[21] P. Akl and A. Moshovos, “Branchtap: improving performance with very few checkpoints through adaptive speculation control,” in *ICS ’06: Proceedings of the 20th annual international conference on Supercomputing*, New York, NY, USA, 2006, pp. 36–45.

[22] D. Brooks, V. Tiwari, and M. 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.