Confidence Estimation for Branch Prediction Reversal

Juan L. Aragón[1], José González[1], José M. García[1] and Antonio González[2]

[1]Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30071 Murcia (Spain)
{jlaragon,joseg,jmgarcia}@ditec.um.es
[2]Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya, 08034 Barcelona (Spain)
antonio@ac.upc.es

Abstract. Branch prediction reversal has been proved to be an effective alternative approach to dropping misprediction rates by means of adding a Confidence Estimator to a correlating branch predictor. This paper presents a *Branch Prediction Reversal Unit* (*BPRU*) especially oriented to enhance correlating branch predictors, such as the *gshare* and the Alpha 21264 metapredictor. The novelty of this proposal lies on the inclusion of data values in the confidence estimation process. Confidence metrics show that the *BPRU* can correctly tag 43% of branch mispredictions as low confident predictions, whereas the *SBI* (a previously proposed estimator) just detects 26%. Using the *BPRU* to reverse the *gshare* branch predictions leads to misprediction reductions of 15% for the SPECint2000 (up to 27% for some applications). Furthermore, the *BPRU+gshare* predictor reduces the misprediction rate of the *SBI+gshare* by an average factor of 10%. Performance evaluation of the *BPRU* in a superscalar processor obtains speedups of up to 9%. Similar results are obtained when the *BPRU* is combined with the Alpha 21264 branch predictor.

1 Introduction

One of the main characteristics of each new processor generation is an increase in the complexity and in the size of the hardware devoted to predict branches. Two main factors motivate the necessity of accurate branch predictors:

- In order to exploit more ILP, processors inspect an increasing number of instructions in each cycle. In such processors, the fetch engine must provide instructions from the correct path at a high rate. To meet these requirements, complex instruction caches and powerful branch prediction engines are needed.
- Reducing the clock cycle has been traditionally used to improve the performance of processors. On a given technology, fewer gates per pipeline stage result in higher frequencies, leading to longer pipelines. This is a challenge for superscalar processors since the branch misprediction penalty is increased.

Branch predictor accuracy can be improved by augmenting its size or complexity. However, traditional misprediction rate figures show that, reached a certain size, the misprediction rate is scarcely reduced. *Confidence Estimation* has become an interesting approach to increasing prediction accuracy by means of assessing the quality of branch predictions [8][10]. The confidence estimator generates a binary signal indicating whether the branch prediction can be considered high or low confidence.

This paper presents a confidence estimator used to reverse the low confidence branch predictions (*Branch Prediction Reversal Unit*). The novelty of this estimator is the inclusion of data values to assign confidence to branch predictions, attempting to use different information from that employed by the branch predictor (usually branch history and branch PC). This can be very efficient in different scenarios such as branches that close loops, list traversals or pathological *if-then-else* structures, where the involved branches may not be correlated with previous history but may be correlated with data values. The main contributions of this paper are:

- We propose a generic *BPRU* that can be used in conjunction with any branch predictor. The *gshare* [14] and the 21264 branch predictor [11] are used as test cases.
- The *BPRU* is compared with a previously proposed confidence estimator, the *SBI* [13]. We first use the metrics proposed in that work and show that including data values as additional information to assign confidence, leads to better confidence estimation. We also compare both estimators for branch prediction reversal.
- We evaluate the impact of adding the *BPRU* to traditional branch predictors in a superscalar processor, and compare it with the impact of adding the *SBI*.

The rest of this paper is organized as follows. Section 2 presents the motivation of our proposal and related work. The proposed *BPRU* is described in Section 3. Section 4 analyzes the accuracy of the *BPRU* in terms of misprediction rate, as well as its performance in terms of IPC. Finally, Section 5 summarizes the conclusions of this work.

2 Related Work and Background

Different branch prediction strategies have been presented to improve prediction accuracy. Most of previous proposals correlate the behavior of a branch with its own history [16][17], the history of previous branches [14], or the path [15]. These initial approaches were later extended with anti-aliasing techniques [3][12] and hybrid predictors [4][14]. Data values have been also incorporated in the branch prediction process. Heil *et al* [9] use the history of data values to access branch prediction tables. Value prediction was directly used to compute the branch outcome in [7], by predicting the inputs of branches and compare instructions and speculatively compute the branch outcome in some dedicated functional units.

Assigning confidence to branch predictions appeared as an orthogonal way of improving accuracy. Jacobsen *et al* [10] propose different schemes to calculate the likelihood of a branch misprediction. They suggest that confidence estimation can be applied to improve prediction accuracy by means of prediction reversal, although no particular implementations are presented. Grunwald *et al* [8] propose different confidence estimators and Manne *et al* [13] evaluate their usefulness to *Selective Branch Inversion*. In [1], we present a branch prediction confidence estimator especially tuned to improve the *Branch Predictor through Value Prediction* proposed in [7]. This confidence estimator relies on the fact that many value mispredictions are concentrated on a small range of data values. Detecting such critical values and reversing those branch predictions made according with them improves the *BPVP* accuracy.

The work we present here differs from previous proposals in the following: a) branch confidence estimators presented in [8][13] are based on branch history and branch PC, whereas our confidence estimator is also based on data values; and b) with respect to our initial proposal [1], we extend here its functionality to make it suitable for any branch predictor, instead of just focusing on improving the *BPVP*.

In order to compare different confidence estimators, Grunwald *et al* [8] defined four metrics: *Sensitivity* (SENS) represents the fraction of correct branch predictions identified as high confidence; *Predictive Value of a Positive Test* (PVP) identifies the probability that a high confidence estimation is correct; *Specificity* (SPEC) is the fraction of incorrect predictions identified as low confidence; and *Predictive Value of a Negative Test* (PVN) is the fraction of low confidence branches that are finally mispredicted. For the application of branch reversal, we are interested in the last two metrics: SPEC and PVN, obtaining a significant potential only if both metrics are high.

3 Branch Prediction Reversal Mechanism

3.1. Quantitative Analysis of the Branch Reversal Benefit

We have first performed an off-line analysis in order to gain some insight into the processor parameters that provide a better knowledge of branch mispredictions. We have examined the following parameters:

1. The predicted value of the branch input.
2. The predicted value of the branch input and the branch PC.
3. The predicted value of the branch input and the PC of the branch input producer.
4. The predicted value of the branch input, the PC of the branch input producer and the recent path followed to reach the branch.
5. The predicted value of the branch input, the PC of the branch input producer and the recent history of branch outcomes.

We have run some benchmarks from the SPECint2000 using a modified version of the *sim-safe* simulator [2]. The number of branch hits and misses of a baseline branch predictor have been measured for all the above combinations of processor state parameters, assuming unbounded storage resources. For those parameter instances whose number of branch mispredictions is greater than the number of hits, the prediction is reversed. Thus, a new misprediction rate is obtained, showing the potential of reversing the branch prediction when considering this *a priori* information.

Fig. 1 shows the new misprediction rate obtained for *bzip2*, *eon*, *mcf* and *twolf* for the five evaluated scenarios. We have used the *Stride Value Predictor* with an unrealistic size of 512 KB trying to isolate the potential of our proposal from the performance of the value predictor. The underlying branch predictor is a 32 KB *gshare*. It can be observed that using the predicted value of the branch input together with the PC of the branch input producer and the history of branch outcomes is the best approach. These results indicate the potential of branch reversal but are not an upper-bound since they have been obtained assuming that each instance of the chosen parameters can be reversed either always or never. In practice, the proposed mechanism will be able to dynamically reverse the prediction only for a certain part of the execution.

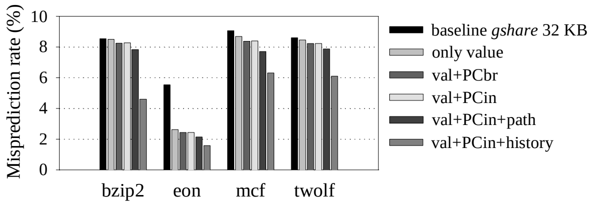

Fig. 1. Potential misprediction rate for a 32 KB *gshare*.

3.2. Branch Prediction Reversal Unit (*BPRU*)

This section presents the implementation of the *Branch Prediction Reversal Unit (BPRU).* This unit could be included in any branch predictor, although in this work we have used the *BPRU* along with the *gshare* and the alpha 21264 branch predictors.

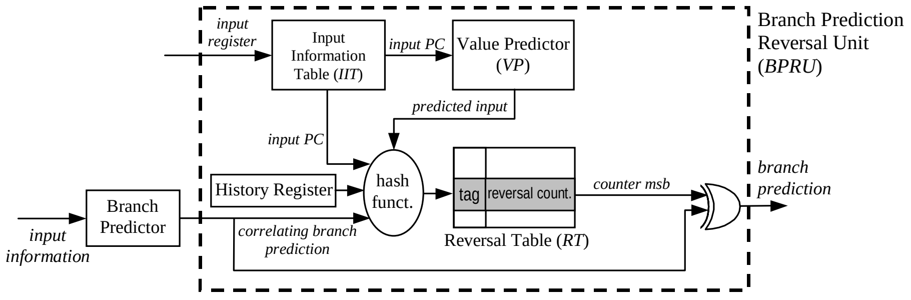

Fig. 2. Block diagram of the *Branch Prediction Reversal Unit (BPRU).*

Fig. 2 depicts a diagram of the *BPRU.* It is composed of an *Input Information Table* (*IIT*), a *Value Predictor* (*VP*), a *History Register* (*HR*) and a *Reversal Table* (*RT*). The *IIT* is a simplified version of the one proposed in [7][1]. The *Value Predictor* can be any known predictor, although in this work we have assumed the *Stride Value Predictor* (*STP*) because of its effectiveness [5][7]. Therefore, the *VP* provides either the predicted branch input value for load/arithmetic/logical instructions or the predicted difference between both inputs for compare instructions. The *History Register* collects the outcomes of recent executed branches. Finally, the confidence for each conditional branch is estimated by the *Reversal Table* (*RT*).

Each entry of the *RT* stores an up-down saturating counter that determines whether the prediction must be reversed, and a tag. The *RT* is accessed when the branch is predicted, by hashing some processor state information. According to the analysis of the previous section, the most effective approach to reversing branch predictions is to

[1] The *IIT* has the same number of entries as total logical registers. Each *IIT* entry only has one field: the PC of the latest instruction that had this register as destination. During decoding, load/arithmetic/logical instructions store their PC in the corresponding entry of the *IIT*, whereas compare instructions first access the *IIT* to obtain the PC of the producers of both inputs, and then store a *xor*-hashed version of both PCs.

correlate with the predicted value, the PC of the branch input producer and the history of recent branch outcomes. In addition, the prediction bit of the baseline predictor is used to index the *RT*, since it is useful for reducing interference in the confidence estimator [7][13]. The most significant bit of the counter of the *RT* entry indicates whether the branch prediction must be reversed. Once the actual branch outcome is computed, the *RT* entry is updated, incrementing the counter if the prediction was incorrect, and decreasing the counter otherwise.

Conflicts in the *RT* are one of the major problems that may limit the accuracy of the *BPRU*. Experimental results showed that a tagged *RT* provides higher accuracy than a non-tagged *RT* of the same total storage, despite the space occupied by tags. Besides, the replacement policy of the *RT* has to be carefully selected. In our case, it gives priority to entries with lower values in their counters.

Since the *BPRU* has to perform three table accesses (*IIT*, *VP* and *RT*) to provide the prediction, its latency may be higher than the latency of the correlating branch predictor. Thus, during the fetch stage the correlating predictor provides its prediction as the initial one. Later, if the *BPRU* changes the initial prediction, the fetched instructions after the branch are flushed and the fetch is redirected to the new PC.

4 Experimental Results

This section analyzes the performance of the proposed *BPRU* engine when it is integrated into the *gshare* and the Alpha 21264 [11][2] branch predictors. For comparison purposes we have also implemented the *SBI* confidence estimator [13]. All experiments compare predictors with the same total size, including the space occupied by tags, counters and any other required storage. We have considered the whole SPECint2000 suite for the evaluation of our proposed mechanism. All benchmarks were compiled with maximum optimizations by the Compaq Alpha compiler, and they were run using the *SimpleScalar/Alpha* v3.0 tool set [2]. We have reduced the input data set of the SPEC2000 suite while trying to keep a complete execution for every benchmark, as showed in Table 1.

Table 1. Benchmark characteristics.

Benchmark	Input Set	Total # dyn. Instr. of Input Set (Mill.)	Total # simulated Instr. (Mill.)	# skipped Instr (Mill.)	# dyn.cond. branch (Mill.)
bzip2	input source 1	2069	500	500	43
crafty	test (modified)	437	437	-	38
eon	kajiya image	454	454	-	29
gap	test (modified)	565	500	50	56
gcc	test (modified)	567	500	50	62
gzip	input.log 1	593	500	50	52
mcf	test	259	259	-	31
parser	test (modified)	784	500	200	64
twolf	test	258	258	-	21
vortex	test (modified)	605	500	50	51
vpr	test	692	500	100	45

[2] The Alpha 21264 processor uses a branch predictor composed by a metapredictor that chooses between the predictions made by a global and a local predictor.

4.1. Confidence Estimator Evaluation

Before analyzing the effect of our *BPRU* on branch prediction accuracy, we evaluate its usefulness as a confidence estimator. The total size of the *BPRU+gshare* is 36 KB (16 KB for the *RT*, 16 KB for the *gshare* and 4 KB for the value predictor) whereas the total size of the *SBI+gshare* is 32 KB (16 KB for the confidence estimator and 16 KB for the *gshare*). Table 2 shows that for both estimators, SENS and PVP metrics are quite similar and very high, which means that correct branch predictions are correctly estimated by both schemes. PVN, is also very similar for both schemes, i.e., 70% of branches estimated as "low confidence" finally miss the prediction. This metric must be greater than 50% in order to a get a positive inversion benefit. Finally, we can observe that the *BPRU* estimator provides a better accuracy for the SPEC metric. On average, 43% of incorrect predictions are identified as low confidence by our estimator, whereas just 26% of them are identified as low confidence by the *SBI*. This metric gives an insight about the "quality" of the confidence estimation: the *BPRU* can detect branch mispredictions much better than the *SBI*.

Table 2. Confidence estimation metrics for the *BPRU+gshare* and the *SBI+gshare*.

Benchmark	BPRU+gshare				SBI+gshare			
	Spec	PVN	Sens	PVP	Spec	PVN	Sens	PVP
bzip2	53,0%	65,9%	97,7%	96,1%	24,1%	62,3%	98,8%	93,9%
crafty	45,7%	65,4%	98,3%	96,3%	28,1%	77,4%	99,4%	95,2%
eon	19,6%	81,3%	99,7%	95,6%	18,2%	61,2%	99,3%	95,5%
gap	44,7%	74,1%	99,4%	97,9%	25,1%	76,3%	99,7%	97,2%
gcc	47,9%	80,3%	99,3%	96,9%	39,0%	82,3%	99,5%	96,4%
gzip	35,1%	62,9%	99,2%	97,5%	18,3%	65,3%	99,6%	96,9%
mcf	36,2%	62,7%	98,3%	95,1%	17,9%	57,8%	99,0%	93,8%
parser	35,7%	67,6%	98,9%	95,9%	29,5%	73,2%	99,3%	95,6%
twolf	41,7%	66,6%	97,8%	94,0%	20,7%	67,0%	98,9%	92,1%
vortex	71,4%	89,5%	99,9%	99,6%	53,0%	89,3%	99,9%	99,4%
vpr	39,0%	55,5%	97,6%	95,4%	16,6%	53,7%	98,9%	93,9%
AVERAGE	42,7%	70,2%	98,7%	96,4%	26,4%	69,6%	99,3%	95,4%

4.2. Results for Immediate Updates

The first set of experiments update prediction tables immediately, in order to evaluate the potential of the selective reversal mechanism when it is isolated from other aspects of the microarchitecture (using the *sim-safe* simulator). The *RT* is implemented as an 8-way set-associative table using 13 bits for tags and 3 bits for the reversal counters. A *RT* index of *k*+1 bits is the result of *XORing*: *k* bits from the PC of the branch input producer, 4 bits of history, *k*-4 bits from the predicted value and finally, the *gshare* prediction bit as the most significant bit.

Fig. 3 shows the misprediction rates for the *BPRU+gshare*, *SBI+gshare* and *gshare* predictors for five selected SPECint2000 programs (those with the highest misprediction rate) and the arithmetic mean of the whole SPECint2000. Based on experimental results, the *BPRU+gshare* allocates 1/9 of the size for the value predictor, 4/9 for the *RT* and 4/9 for the *gshare*. The *SBI+gshare* allocates 1/2 for the confidence estimator and 1/2 for the *gshare*.

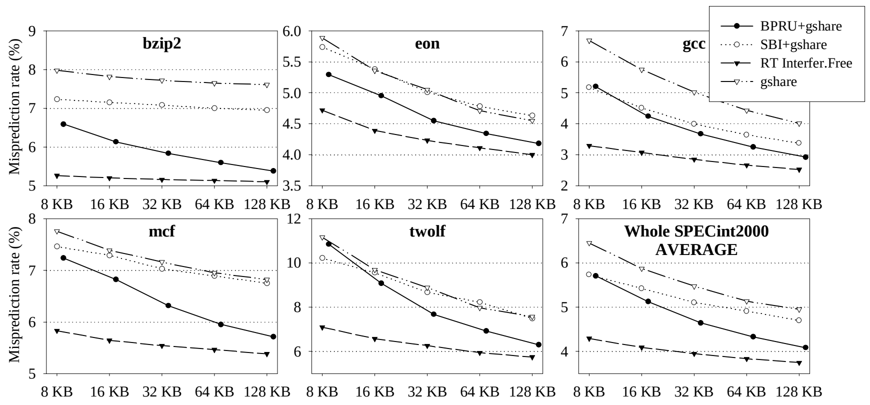

Fig. 3. Branch misprediction rates for the *BPRU+gshare*, *SBI+gshare* and *gshare* predictors.

The *BPRU+gshare* predictor significantly reduces the misprediction rate of the *gshare* of the same total capacity for all benchmarks and all evaluated sizes. For predictors of around 32 KB, the misprediction rate of the *gshare* is reduced by an average of 15% (up to 27% for *gcc*). It is important to note that the *BPRU+gshare* with a total size of 18 KB has the same misprediction rate (5.1%) as the *gshare* of 64 KB.

Comparing the *BPRU* and the *SBI* schemes, the *BPRU+gshare* outperforms the *SBI+gshare* for all benchmarks for sizes greater than 8 KB. On average, the *BPRU+gshare* with a total size of 36 KB reduces 10% the misprediction rate of the *SBI+gshare* of 32 KB (up to 18% for *bzip2*). Summarizing, for the whole SPECint2000 suite, the *BPRU+gshare* reduces the misprediction rate by a factor that ranges from 12% (8 KB) to 18% (128 KB) with respect to the *gshare*, and from 5% (16KB) to 13% (128 KB) with respect to the *SBI+gshare*.

Finally, Fig. 3 shows that the interference-free *RT* provides great improvements for all benchmarks. For instance, in the *twolf* application, the misprediction rate of an 8 KB *gshare* is reduced by a factor of 37%. This shows the potential of the *BPRU* and an opportunity for improvement by using better indexing schemes to access the *RT*.

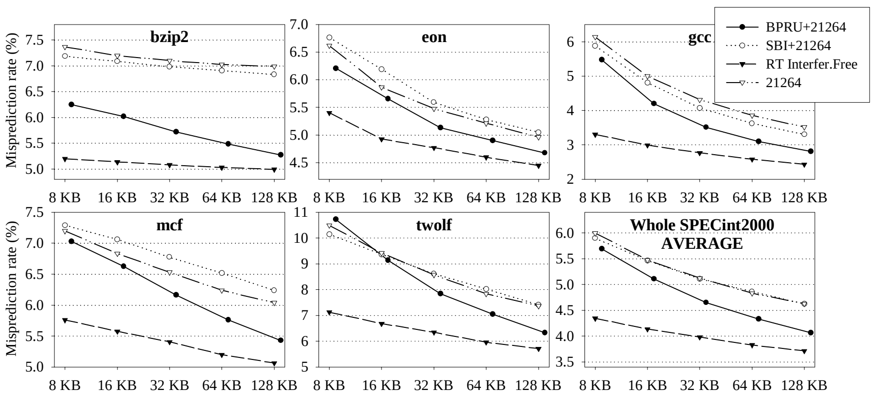

Fig**. 4.** Branch misprediction rates for the *BPRU+21264*, *SBI+21264* and *21264* predictors.

Fig. 4 shows the same analysis when the 21264 branch predictor is the underlying predictor. For 32 KB predictors, adding the *BPRU* reduces misprediction rate by an average of 9% (up to 19% for *bzip2* and *gcc*). We can also observe that, for these applications, the *SBI* hardly provides benefits. Regarding predictor sizes, the *BPRU+21264* of 36 KB obtains a similar misprediction rate as the 21264 branch predictor of 128 KB. Summarizing, average reductions in the misprediction rate provided by the *BPRU+21264* range from 5.1% (8 KB) to 12.1% (128 KB) with respect to the 21264 branch predictor, and from 3.5% (8 KB) to 12.0% (128 KB) over the *SBI+21264*.

Results presented in this Section show the usefulness of the *BPRU* as a confidence estimator for branch prediction reversal, demonstrating that adding the *BPRU* to a correlating branch predictor is more effective than simply increasing its size.

4.3. Results for Realistic Updates

This Section presents an evaluation of the proposed *BPRU* engine in a dynamically-scheduled superscalar processor. Table 3 shows the configuration of the simulated architecture. In addition, the original *SimpleScalar* simulator pipeline has been lengthened to 20 stages, following the pipeline scheme of the Pentium 4 processor [6].

Table 3. Configuration of the simulated processor.

Fetch engine	Up to 8 instructions/cycle, allowing 2 taken branches. 8 cycles misprediction penalty.
Execution engine	Issues up to 8 instructions/cycle, 128-entries reorder buffer, 64-entries loads/store queue.
Functional Units	8 integer alu, 2 integer mult, 2 memports, 8 FP alu, 1 FP mult.
L1 Instruct.-cache	128 KB, 2-way set associative, 32 bytes/line, 1 cycle hit latency.
L1 Data-cache	128 KB, 2-way set associative, 32 bytes/line, 1 cycle hit latency.
L2 unified cache	512 KB, 4-way set associative, 32 bytes/line, 6 cycles hit latency, 18 cycles miss latency.
Memory	8 bytes/line, virtual memory 4 KB pages, 30 cycles TLB miss.

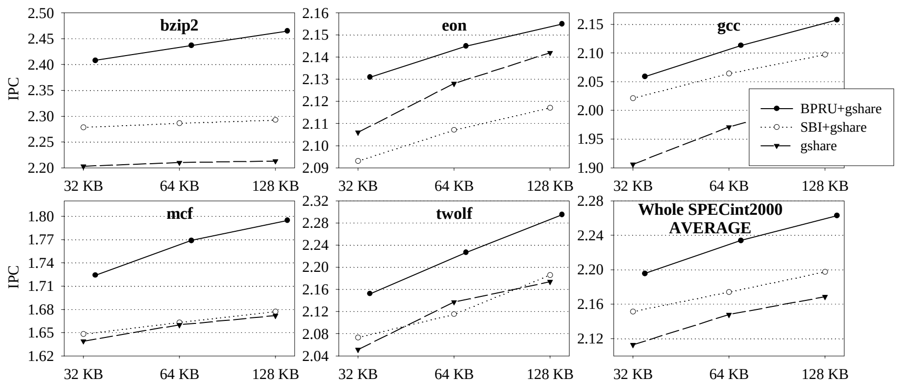

Fig. 5. IPC obtained by the *BPRU+gshare*, *SBI+gshare* and *gshare* branch predictors.

Fig. 5 shows the IPC obtained by the simulated processor when it incorporates the *gshare*, *SBI+gshare* and *BPRU+gshare*. The latency of both the *gshare* and the

SBI+gshare is considered to be one cycle, whereas the latency considered for the *BPRU+gshare* is 3 cycles. For 32 KB predictors, and in spite of its higher latency, the *BPRU+gshare* provides performance improvements over the *gshare* of 9% and 8% for *bzip2* and *gcc* respectively. As the size of the predictors grows, performance improvements are also increased, up to 11% for *bzip2* (128 KB). Comparing both confidence estimators, the *BPRU+gshare* outperforms the *SBI+gshare* for all programs and predictor sizes, obtaining speedups of up to 6% for 32 KB predictors.

Fig. 6 presents performance results using the underlying 21264 branch predictor. For 32 KB predictors, the *BPRU+21264* outperforms the 21264 branch predictor by 7% and 5% for *bzip2* and *gcc*. Comparing both confidence estimators, the *BPRU+ 21264* obtains speedups of up to 6% over the *SBI+21264*, for 32 KB predictors. These speedups are similar to those obtained when the *gshare* is the baseline predictor.

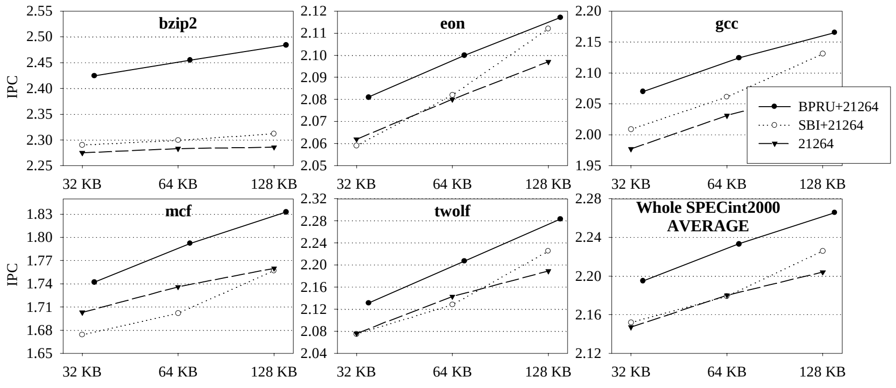

Fig. 6. IPC obtained by the *BPRU+21264*, *SBI+21264* and 21264 branch predictors.

5 Conclusions

In this work, we have presented a new branch prediction confidence estimator, the *Branch Prediction Reversal Unit (BPRU).* The key characteristic of this estimator is the inclusion of predicted data values in the process of assigning confidence to the predictions. We have shown that the accuracy of conventional correlating predictors such as the *gshare* and the Alpha 21264 branch predictor can be effectively correlated with the predicted data value, the PC of the producers of the branch input and recent branch history. We have first analyzed the *BPRU* as a confidence estimator, comparing it with an already proposed scheme, the *SBI.* Confidence metrics show that the *BPRU* is able to label as “low confidence” 43% of branch mispredictions, conversely to the *SBI*, which just assigns low confidence to 26% of branch mispredictions.

We have also evaluated the *BPRU* as a branch prediction reversal mechanism for the *gshare* and the 21264 branch predictors. Misprediction reductions are quite similar for both correlating predictors. Results for immediate updates of the prediction tables show misprediction reductions of 15% on average for the whole SPECint2000 suite (up to 27% for some applications) when 32 KB predictors are considered. Comparing the *BPRU* and the *SBI*, we show that, for 32 KB predictors, the *BPRU+ gshare* reduces the misprediction rate of the *SBI+gshare* by an average of 10%. Finally, performance evaluation of the *BPRU* in a superscalar processor show IPC improvements of up to 9% over both the *gshare* and the *SBI+gshare*, using 32 KB predictors. To conclude, the *BPRU* is a cost-effective way of improving branch prediction accuracy, being more effective than simply increasing the size of the predictor.

Acknowledgements

This work has been supported by the project TIC2000-1151-C07-03 of the *Plan Nacional de I+D+I* and by the Spanish CICYT under grant TIC98-0511.

References

1. Aragón, J.L., González, J., García, J.M., González, A.: Selective Branch Prediction Reversal by Correlating with Data Values and Control Flow. In: Proceedings of the Int. Conf. on Computer Design. (2001)
2. Burger, D., Austin, T.M.: The SimpleScalar Tool Set, Version 2.0. Technical Report #1342, University of Wisconsing-Madison, Computer Sciences Department. (1997)
3. Chang, P.Y., Evers, M., Patt, Y.N.: Improving Branch Prediction Accuracy by Reducing Pattern History Table Interference. In: Proceedings of the Int. Conf. on Parallel Architectures and Compilation Techniques. (1996)
4. Chang, P.Y., Hao, E., Patt., Y.N.: Alternative implementations of Hybrid Branch Predictors. In: Proceedings of the Int. Symp. on Microarchitecture. (1995)
5. Gabbay, F., Mendelson, A.: Speculative Execution Based on Value Prediction. Technical Report #1080, Technion, Electrical Engineering Department. (1996)
6. Glaskowsky, P.N.: Pentium 4 (Partially) Previewed. Microprocessor Report, Microdesign Resources. (August 2000)
7. González, J., González, A.: Control-Flow Speculation through Value Prediction for Superscalar Processors. In: Proc. of the Int. Conf. on Parallel Arch. and Comp. Tech. (1999)
8. Grunwald, D., Klauser, A., Manne, S., Pleszkun, A.: Confidence Estimation for Speculation Control. In: Proceedings of the Int. Symp. on Computer Architecture. (1998)
9. Heil, T.H., Smith, Z., Smith, J.E.: Improving Branch Predictors by Correlating on Data Values. In: Proceedings of the Int. Symp. on Microarchitecture. (1999)
10. Jacobsen, E., Rotenberg, E., Smith, J.E.: Assigning Confidence to Conditional Branch Predictions. In: Proceedings of the Int. Symp. on Microarchitecture. (1996)
11. Kessler, R.E., McLellan, E.J., Webb, D.A.: The Alpha 21264 Microprocessor Architecture. In: Proceedings of the Int. Conf. on Computer Design. (1998)
12. Lee, C.C., Chen, I.C.K., Mudge, T.N.: The Bi-Mode Branch Predictor. In: Proceedings of the Int. Symp. on Microarchitecture. (1996)
13. Manne, S., Klauser, A., Grunwald, D.: Branch Prediction using Selective Branch Inversion. In: Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques. (1999)
14. McFarling, S.: Combining Branch Predictors. Technical Report #TN-36, Digital Western Research Lab. (1993)
15. Nair, R.: Dynamic Path-Based Branch Correlation. In: Proceedings of the Int. Symp. on Microarchitecture. (1995)
16. Smith, J.E.: A Study of Branch Prediction Strategies. In: Proceedings of the Int. Symp. on Computer Architecture. (1981)
17. Yeh, T.Y., Patt, Y.N.: A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History. In: Proceedings of the Int. Symp. on Computer Architecture. (1993)