Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Systems Architecture 52 (2006) 143–151
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

Improved composite confidence mechanisms for a perceptron branch predictor

Veerle Desmet *, Lieven Eeckhout, Koen De Bosschere

Ghent University—UGent, Department of Electronics and Information Systems (ELIS), Parallel Information Systems (PARIS) Group, member HiPEAC, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium

Received 11 April 2003; received in revised form 24 March 2005; accepted 13 April 2005
Available online 12 July 2005

Abstract

In 2001, Jiménez and Lin [Dynamic branch prediction with perceptrons, Proceedings of the 7th International Symposium on High Performance Computer Architecture, 2001, pp. 197–206] introduced the perceptron branch predictor, the first dynamic branch predictor to successfully use neural networks. This simple neural network achieves higher accuracies (95% at a 4 KiB hardware budget) compared to other existing branch predictors and provides a free confidence level. In this paper, we first gain insight into this inherent confidence mechanism of the perceptron predictor and explain why (additional) counter based confidence strategies can complement it. Second, we evaluate several composite confidence estimation strategies and compare them to the described technique by Jiménez and Lin [Composite confidence estimators for enhanced speculation control, Tech. rep., Department of Computer Sciences, The University of Texas at Austin, 2002]. We conclude that our *overruling AND-combination* of perceptron confidence and resetting counter mechanism outperforms the previously proposed confidence scheme.

Keywords: Computer architecture; Branch prediction; Neural network; Perceptron predictor; Confidence mechanism

1. Introduction

Deeply pipelined superscalar microarchitectures as we know them today rely on a fetching mechanism that provides several useful instructions every clock cycle. Conditional branch instructions cause difficulties in this constant feeding process: the next instruction to be executed is not known until the branch condition (e.g., argument equals zero)

* Corresponding author. Tel.: +32 9 264 3594; fax: +32 9 264 3405.
E-mail addresses: veerle.desmet@elis.ugent.be (V. Desmet), lieven.eeckhout@elis.ugent.be (L. Eeckhout), koen.debosschere@elis.ugent.be (K. De Bosschere).

1383-7621/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2005.04.003

is computed. This computation typically completes 3–14 cycles after the branch has entered the pipeline. In the meanwhile no further instructions can be fetched.

To solve this problem, modern microarchitectures rely on *branch prediction*, which operates in the fetch unit. A branch predictor predicts the outcome of the branch condition so that instructions on the predicted path can enter the pipeline the next cycle. This enables a constant feeding to the pipeline but involves additional complications for handling speculative instructions and verifying the prediction. Of course the branch instruction itself must be executed to verify the prediction. On a correct prediction all speculative instructions are useful and finish earlier compared to no branch prediction. On a misprediction however, all speculative work has to be undone before the correct path can be executed and this results in even an extra penalty compared to no branch predictor.

As pipelines deepen and the number of instructions issued per cycle increases, the penalty for a misprediction also increases. Current branch predictors reach 95% prediction accuracy and continuous improvements in new directions are exploited in this domain. The driving force behind these efforts is the large improvement in the average number of instructions executed per cycle (IPC) that is possible by even a small improvement in prediction accuracy [3].

Although speculative execution is essential for increasing the IPC, unnecessary work enters the pipeline due to branch mispredictions, resulting in e.g., an increased power consumption [4]. As such, to limit the amount of power consumed it is better not to predict than to predict incorrectly. Therefore assigning a degree of reliability or *confidence* to each dynamic branch is important for balancing the benefits of speculative work. To assign confidence to branches most branch predictors need to keep additional confidence information. In this paper, we focus on the perceptron branch predictor where useful confidence information is for free.

This paper is organized as follows: Section 2 gives an overview of previous work done in branch prediction and confidence mechanisms, while Section 3 describes the perceptron predictor as proposed by Jiménez and Lin. In Section 4, confidence mechanisms for branch prediction and metrics for comparing different confidence strategies are discussed. Section 5 describes our methodology, Section 6 explains the details about the perceptron confidence, and Section 7 points out that perceptron confidence can be complemented by resetting counters. In Section 8 composite mechanisms are compared against previously proposed strategies and we show that our mechanism performs better. Section 9 summarizes this paper.

2. Related work

During the last 15 years, much research in the domain of branch prediction was done to improve branch prediction accuracy. Important contributions in the area of dynamic branch prediction include [5–10].

Calder et al. [11] used neural networks for static branch prediction (prediction at compile time or a fixed prediction per static branch). Their technique achieves much higher static prediction accuracy than previous work, but performs worse than dynamic techniques.

In 2001, Jiménez and Lin [1] introduced the perceptron branch predictor, the first dynamic branch predictor to successfully use neural networks. This simple neural network achieves higher accuracies compared to other existing branch predictors and provides a free confidence level.

In [12], Jacobsen et al. introduce the concept of confidence to improve the effectiveness of branch prediction by concentrating as many of the mispredictions into a small set of low confidence branches. Confidence mechanisms assign *high confidence* or *low confidence* such that assigning high confidence goes together with small chance of making a misprediction. High-confident predictions will be used whereas for low-confident ones behave as if no branch predictor is available, meaning that the pipeline stalls until the branch outcome is computed.

Manne et al. describe the need of speculation control through confidence estimation and show the effect in terms of energy reduction [4] and Grunwald et al. propose metrics for comparing different confidence strategies in [13].

A study by Jiménez and Lin [2] focuses on composite confidence estimators that are useful in wide ranges. We will show that our composite confidence strategies outperform the one proposed in the work by Jiménez and Lin.

3. Perceptron predictor

In 2001, Jiménez and Lin introduced a perceptron branch predictor [1]. As simple neural network, a single layer perceptron model learns to classify examples in two classes. Fig. 1 illustrates the principle of a perceptron branch predictor. The perceptron output y serves to decide whether the branch is predicted taken ($y \geqslant 0$) or not-taken ($y < 0$). This perceptron output y is computed as the dot product of the weight vector with the input vector. The weight vector, $[W_0, \ldots, W_h]$, is taken from the perceptron table indexed by the branch address. The input vector, $[X_0, \ldots, X_h]$ consists of global history information—outcomes of previously resolved branches—of length h, and one bias X_0 which is always set to 1. Thus, the perceptron output y equals:

$$y = W_0 \cdot 1 + \sum_{i=1}^{h} (W_i \cdot X_i)$$

In the previous expression, we encode each input bit X_i as either 1 or -1 for a global history bit corresponding to a taken or not-taken branch, respectively.

When the actual branch outcome is known, the used weight vector is updated. During this update, each weight is incremented/decremented when the branch outcome agrees/disagrees with the corresponding input bit, respectively. Consequently, the stronger the correlation, i.e. agreement or disagreement, the larger the weight and the higher its contribution to the perceptron output y. As reported by Jiménez and Lin [1], weights are only updated on a misprediction or when y holds a small value ($|y| \leqslant \lfloor 1.93h + 14 \rfloor$, with h the length of the global history). In the latter case the perceptron output is considered poorly convincing and further learning is performed to obtain a more robust prediction in the future. A single layer perceptron is easy to understand but deals with the limitation that only so-called linearly separable functions can be learned perfectly.

4. Confidence mechanisms

A confidence mechanism assigns to each prediction high or low confidence based on some *confidence information*. We first describe saturating counters as a type of confidence information and then we explain the free perceptron confidence. At the end of this section we show how different confidence mechanisms can be compared.

A *saturating counter* directly represents the confidence of the corresponding prediction. If the counter value is lower than a certain *threshold* the prediction is assigned low confidence, otherwise high confidence. The higher the threshold, the stronger the confidence mechanism. Regardless of the assigned confidence, the counter is updated at the moment the computed value is known. For this update we discuss two strategies: up/down counters and resetting counters. Up/down counters [14] increment the counter (by one) for a correct prediction saturating at the maximum counter value and decrement (by one)

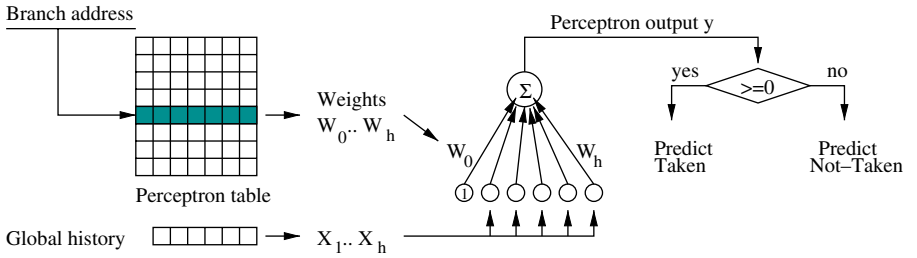

Fig. 1. Schematic working of the perceptron branch predictor.

the counter down to zero on a misprediction. Resetting counters [12] increment (by one) for a correct prediction but reset the counter to zero on a misprediction.

The perceptron predictor provides its own level of confidence. Indeed, the network output is not a Boolean value but a number proportional to the certainty that the branch is taken. In Fig. 2 we illustrate the previous statement with a simulation result for `176.gcc` where the misprediction rate is shown as a function of the perceptron output. This graph clearly shows that a higher misprediction rate is observed for perceptron output values close to zero. Hence the magnitude of the perceptron output, $|y|$, can be used for assigning confidence: high confidence if a certain threshold is crossed and low confidence otherwise. This confidence mechanism in the perceptron predictor is for free whereas other branch predictors and confidence techniques explicitly have to use additional storage to keep confidence information.

For comparing confidence mechanisms, we use the independent metrics proposed by Grunwald et al. [13]. *Predictive value for a positive test*, shortly *PVP*, represents the probability that a high confidence prediction is correct, and the *sensitivity* is the fraction of correct predictions identified as high confidence:

$$PVP = \text{Prob}[\text{correct prediction}|\text{high confident}]$$

$$Sensitivity = \text{Prob}[\text{high confident}|\text{correct prediction}]$$

Both metrics are "higher-is-better".

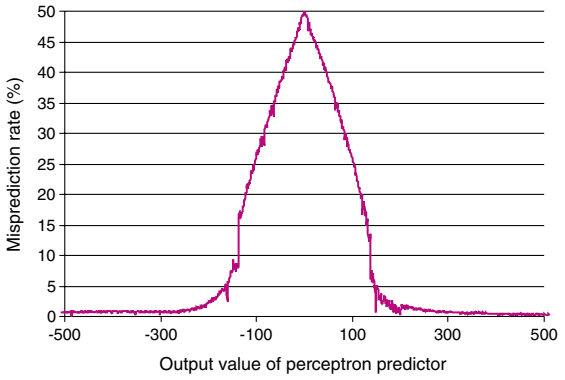

Fig. 2. Misprediction rate versus output of perceptron for `176.gcc`.

5. Evaluation

To obtain our experimental results, we use the SimpleScalar/Alpha simulator (sim-bpred) [15]. We simulate the 12 programs in the SPEC CPU 2000 suite of integer benchmarks which are compiled with the Compaq C compiler version V6.3-025 with optimization flags `-arch ev6 -fast -O4` and Fortran compiler version X5.3-1155 with optimization flags `-arch ev6 -O5`. For all simulations we skip the first 50 million conditional branches and measure for the next 250 million conditional branches. The presented results show the average over all the SPEC benchmarks. When not explicitly mentioned otherwise we use a 3.5 KiB-perceptron predictor that uses a history length of 24 (exclusive bias), a perceptron table of 163 entries and 7 bits for representing a weight (inclusive sign bit). This configuration was found optimal for a total hardware cost lower or equal to 4 KiB in [16] and has a prediction accuracy of 95%.

6. Perceptron confidence

As mentioned in Section 4, the perceptron output can serve as confidence level. In Fig. 3, both the misprediction rate and the number of predictions made by a certain value of the perceptron output are shown. This figure is made for very long history length of 64 together with an infinite perceptron table to avoid the effect of aliasing.

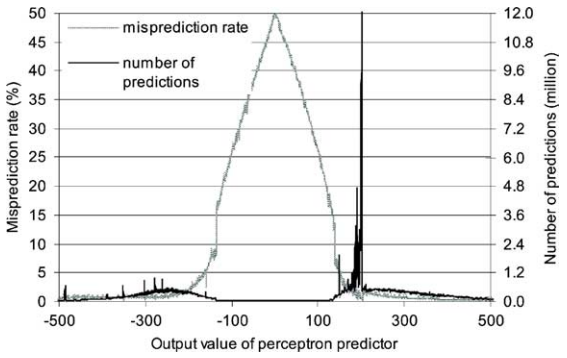

Fig. 3. Misprediction rate (cfr. Fig. 2) and number of predictions versus output of perceptron for `176.gcc`.

First, we note that for `176.gcc` most of the predictions were taken as a positive perceptron output occurred more frequent. Secondly, the figure clearly illustrates that only a small number of predictions is made in the range where the misprediction rate is high. This observation stresses the point that perceptron confidence is useful. We can also see that the range where the threshold for perceptron confidence has to be placed is roughly below 100, which means that branches with output values between −100 and 100 are considered low confident.

6.1. Comparison between perceptron confidence and saturating counters

In Fig. 4 we compare the free confidence mechanism as provided by the perceptron predictor to adding 4-bit saturating counters. In this graph, we show the sensitivity versus the predictive value of a positive test. Recall that both metrics are "higher-is-better" and thus closer to the upper right corner in Fig. 4 is better. The average result over all benchmarks is plotted for a confidence mechanism based on resetting saturating counters (up to 4-bit), for up/down saturating counters (up to 4-bit) and a mechanism based on the perceptron confidence. The common point at the highest possible sensitivity uses all predictions and represents a predictor without confidence (prediction accuracy 95%). All curves are obtained by varying the threshold for assigning high confidence: for the perceptron confidence we augment the threshold up to 100; for the counter mechanisms the threshold is varied from 0 up to 15 (4-bit) and the counter is accessed in parallel with the weights in the perceptron table. First we note that up/down counters are not useful in this case since their results perform worse than the free perceptron confidence over the whole range. For resetting counters we get the expected behavior: every step an improvement for the predictive value of positive test and a decrease in sensitivity. For a range of confidence thresholds, the perceptron confidence technique provides better results over saturating counters; both PVP and sensitivity are higher. However, the perceptron confidence achieves very low sensitivity for higher thresholds.

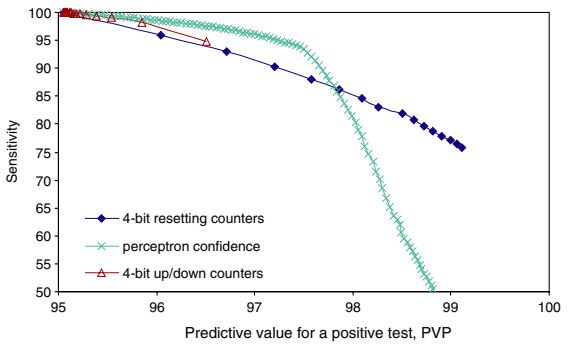

Fig. 4. Comparison between additional confidence mechanism with saturating counters and perceptron confidence.

We observe a knee in the curve at a perceptron threshold of 60 (this corresponds in Fig. 3 with a threshold of 137 because of the history length of 64). At this threshold, which is the one used to decide whether further training of the perceptron is needed, there is a significant increase of both correct and total predictions due to the specific design of the perceptron predictor. Below this threshold, the slope of the curve is better because perceptron confidence benefits from ignoring almost exclusive mispredictions. Above this threshold correct and incorrect predictions are mixed and as confidence estimation becomes harder when prediction accuracy increases [13] in this region the slope is even worse than for resetting counters.

7. Complementary confidence mechanisms

In Fig. 5, the prediction is shown as a function of the perceptron output and the resetting saturating counter state (averaged over all benchmarks). From this figure we see that the higher the (resetting) counter state, the higher the prediction accuracy. An individual confidence strategy considers every element in a row (or column) equivalently and forces the same confidence assignment, while it is clear from Fig. 5b that this is not optimal. We also note that in this sense perceptron confidence can do better than resetting counters. This concludes that perceptron confidence can be complemented by a resetting counter mechanism into a composite confidence estimator.

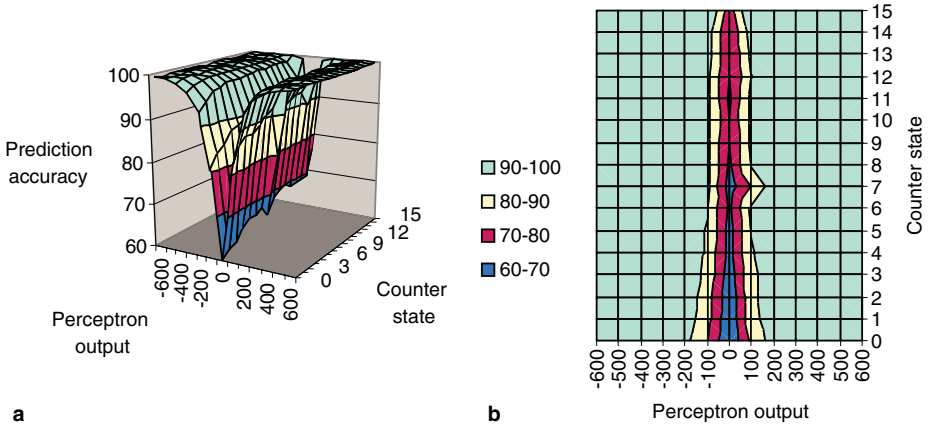

Fig. 5. Correct prediction rate versus perceptron output and saturating counter state. (a) 3D-plot and (b) 2D-projection.

8. Composite confidence mechanisms

We start with the description of an experiment done in other work [2]. For the perceptron predictor they report two combinations: (i) up/down counters and resetting counters and (ii) up/down counters, resetting counters and *self estimation*.[1] The self estimator is the freely provided confidence based on the perceptron output. Instead of using the raw perceptron output they use a shifted version of this value so that it is between 0 and 15.

When confidence mechanisms are combined one has to decide how the individual confidence assignments contribute in the final confidence decision. In [2], the composite confidence mechanism adds up all confidence values of the underlying estimators and this result is compared to a single confidence threshold.

We explore much simpler possibilities including OR-combining and AND-combining of the individual confidence decisions. The first strategy assigns high confidence as soon as one of the underlying mechanisms indicate high confidence, whereas the second strategy only assigns high confidence when both mechanisms agree for high confidence. For the AND-combination it is clear that the sensitivity will achieve lower values as the number of highly-confident correct branches decreases while the total number of correctly predictable branches remains constant (because the predictor is not changed). Reversely, the OR-combination will reach higher sensitivities. A similar reasoning in terms of the predictive value of a positive test is impossible, but a stronger mechanism should avoid more mispredictions than it loses correct predictions so that the PVP increases. In the limit when the sensitivity decreases down to 0% by using none of the predictions, PVP is strictly undefined, but we assume it approaches 100%.

In Fig. 6, the average result is shown for the best composite confidence mechanisms for OR- and AND-combinations compared to the sum-based combination from Jiménez and Lin. First

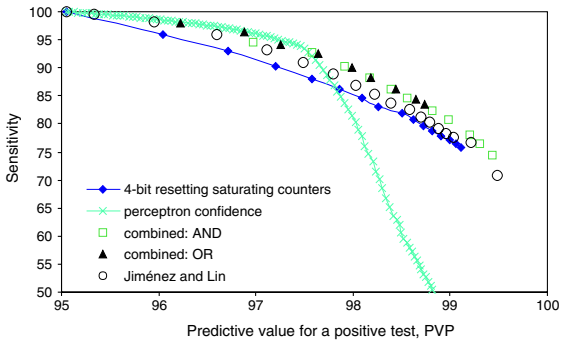

Fig. 6. Combination of perceptron confidence and resetting saturating counter.

[1] Through our experiments, we observed that this combination performs as well as resetting counters and self estimation for the application we envision in this paper.

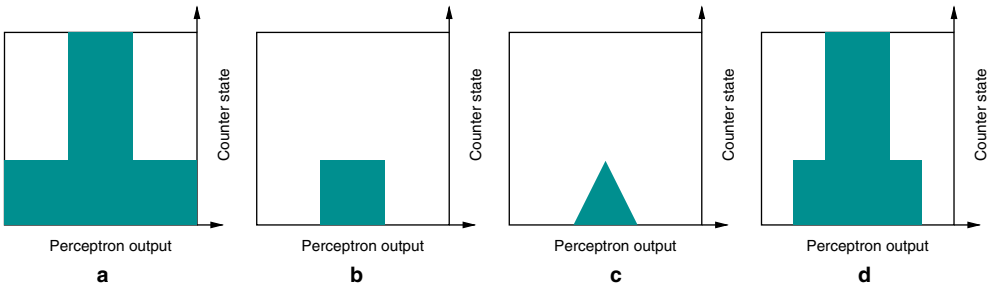

Fig. 7. Confidence assignment in combined confidence mechanisms (dark areas show low-confident branch assignments). (a) and, (b) or, (c) sum-based and (d) overruling.

it is clear that our OR- and AND-combinations outperform the sum-based technique in [2]. The reason for this is that all confidence values are considered of equal importance. However in practice, a high magnitude of the perceptron value is a much better indication for a low misprediction rate than a high saturating counter value. Secondly, the difference between OR- and AND-combinations is extremely small. For the best AND-combinations and as the mechanism becomes stronger (towards the lower right corner) the perceptron threshold is slightly increased but especially the resetting counter threshold is raised up to its maximum value. A similar analysis for the best OR-combinations learns that both confidence threshold vary from moderate to strong.

Thus far, we conclude that resetting counters can complement perceptron confidence to obtain better (composite) confidence estimators and that this can be done by a very simple combination strategy as AND or OR.

Fig. 7(a) shows when low confidence is assigned as a function of the perceptron output and the counter state in the different combination strategies. For the AND-combination technique it appears that for branches that are considered high confident through the perceptron confidence, the final decision depends on the state of the resetting counter. This means that in the major part of the perceptron output range, where it is known that branches are highly accurately predicted, we trust the counter decision. Therefore we propose a solution by adding a second threshold for the perceptron confidence and use this as *overruling threshold*: when the perceptron output reaches this overruling threshold, high confidence overrules the decision of the counter by high confidence (see Fig. 7(d)). The idea behind this strategy is that the saturating counter is valuable in a certain region where the perceptron output alone is not reliable enough, but outside this region the perceptron confidence is more accurate.

In Fig. 8, the average result is shown for the best configurations that use the AND-overruling confidence strategy. These best combinations correspond with the following parameter settings (perceptron threshold, overruling threshold, counter threshold): (25, 35, 3); (35, 60, 3); (45, 75, 3); (35, 75, 9); (55, 80, 9); (55, 90, 9); (55, 100, 9); (55, 100, 15); (65, 110, 9); (65, 110, 15); (65, 120, 15); (75, 105, 6); (75, 105, 12); (75, 135, 9). For these best combinations hold: the overruling threshold is about double the value of the perceptron threshold. This means that for low perceptron thresholds, where the misprediction rate is relative large, the resetting counter is only needed in a

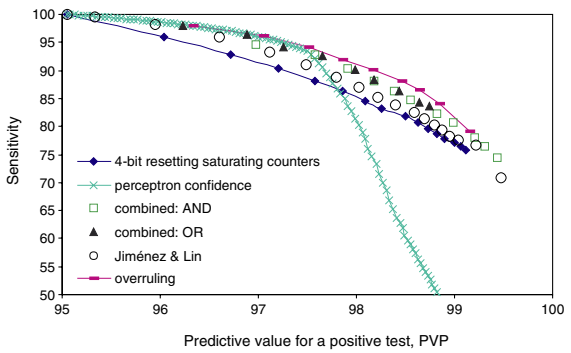

Fig. 8. AND-combination with overruling of the perceptron output outperforms other combination mechanisms.

small range. For higher perceptron thresholds, the resetting counter is useful in a larger range.

Overruling gains over the basic AND-combination because it benefits from the high confidence assignment in a highly accurate perceptron range. The hardware cost (both in time and space) to get the final confidence assignment are from the same complexity as the sum-based technique. As such, we conclude that our overruling AND-combination outperforms all previously described techniques.

9. Conclusion

Although branch prediction is essential for increasing the IPC, unnecessary work enters the pipeline due to branch mispredictions. Therefore confidence techniques are added to balance the amount of speculative work. The introduction of the perceptron predictor (95% at a 4 KiB hardware budget) offers new perspectives in this domain because it provides a free confidence level. In this paper we explored insight into this inherent confidence mechanism of the perceptron predictor and showed that (additional) counter based confidence strategies can complement it. We evaluate several composite confidence estimation strategies and compare them to the described technique by Jiménez and Lin [2]. Especially our overruling AND-combination of perceptron confidence and resetting counter mechanism outperforms the previously proposed sum-based confidence scheme.

Acknowledgements

Veerle Desmet is supported by a grant from the Flemish Institute for the Promotion of Scientific-Technological Research in the Industry (IWT). Lieven Eeckhout is a postdoctoral researcher of the Fund for Scientific Research-Flanders (FWO). This research was also funded by Ghent University.

References

[1] D.A. Jiménez, C. Lin, Dynamic branch prediction with perceptrons, in: Proceedings of the 7th International Symposium on High Performance Computer Architecture, 2001, pp. 197–206.

[2] D.A. Jiménez, C. Lin, Composite confidence estimators for enhanced speculation control, Tech. rep., Department of Computer Sciences, The University of Texas at Austin, 2002.

[3] H. Neefs, K. De Bosschere, J. Van Campenhout, An analytical model for performance estimation of modern data-flow style scheduling microprocessors, in: Proceedings of the 22nd Euromicro Conference: Short Contributions, 1996, pp. 2–7.

[4] S. Manne, A. Klauser, D. Grunwald, Pipeline gating: speculation control for energy reduction, in: Proceedings of the 25th Annual International Symposium on Computer Architecture, 1998, pp. 132–141.

[5] A.N. Eden, T.N. Mudge, The YAGS branch prediction scheme, in: Proceedings of th 31st Annual International Symposium on Microarchitecture, 1998, pp. 69–77.

[6] M. Evers, P.-Y. Chang, Y.N. Patt, Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches, in: Proceedings of the 23rd Annual International Symposium on Computer Architecture, 1996, pp. 3–11.

[7] C.-C. Lee, I.-C.K. Chen, T.N. Mudge, The bi-mode branch predictor, in: Proceedings of the 30th Annual International Symposium on Microarchitecture, 1997, pp. 4–13.

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

[9] E. Sprangle, R.S. Chappell, M. Alsup, Y.N. Patt, The Agree predictor: a mechanism for reducing negative branch history interference, in: Proceedings of the 24th Annual International Symposium on Computer Architecture, 1997, pp. 284–291.

[10] T.-Y. Yeh, Y.N. Patt, Two-level adaptive training branch prediction, in: Proceedings of the 24th Annual International Symposium on Microarchitecture, 1991, pp. 51–61.

[11] B. Calder, D. Grunwald, M. Jones, D. Lindsay, J. Martin, M. Mozer, B. Zorn, Evidence-based static branch prediction using machine learning, ACM Trans. Program Lang. Syst. 19 (1) (1997) 188–222.

[12] E. Jacobsen, E. Rotenberg, J.E. Smith, Assigning confidence to conditional branch predictions, in: International Symposium on Microarchitecture, 1996, pp. 142–152.

[13] D. Grunwald, A. Klauser, S. Manne, A. Pleszkun, Confidence estimation for speculation control, in: Proceedings of the 25th Annual International Symposium on Computer Architecture, 1998, pp. 122–131.

[14] A. Klauser, S. Manne, D. Grunwald, Selective branch inversion: Confidence estimation for branch predictors, Int. J. Parallel Program. 29 (1) (2001) 81–110.

[15] D. Burger, T.M. Austin, S. Bennett, Evaluating future microprocessors: the SimpleScalar Tool Set, Tech. Rep., Computer Sciences Department, University of Wisconsin-Madison, July 1996.

[16] D.A. Jiménez, Neural methods for dynamic branch prediction, ACM Trans. Comput. Syst. 20 (4) (2002) 369–397.

Veerle Desmet was born in Ghent, Belgium in 1978. She received the Engineering degree in Computer Science from Ghent University, Belgium, in 2001. Veerle Desmet is currently working towards a Ph.D. at the Department of Electronics and Information Systems (ELIS) in the same university. For her research, she is sponsored by the Flemish Institute for the Promotion of Scientific-Technological Research in the Industry (IWT). The topic of her research is in the area of computer architecture and more specific in branch prediction.

Lieven Eeckhout was born in Kortrijk, Belgium in 1975. He received the Engineering degree and PhD degree in Computer Science from Ghent University, Belgium, in 1998 and 2002, respectively. Lieven Eeckhout currently is a Postdoctoral Fellow of the Fund for Scientific Research—Flanders (Belgium) (F.W.O.—Vlaanderen). His research interest includes computer architecture, performance analysis and workload characterization.

Koen De Bosschere is a research professor at the Engineering Faculty of the Ghent University where he teaches courses on computer architecture and operating systems. His current research interests are global optimization, performance modelling, microarchitecture, and debugging. He is the head of a research group of 20 researchers. He is the coordinator of the Flemish research network on Architectures and Compilers for Embedded Systems (ACES), and he is the Belgian representative of the HiPEAC network of Excellence.