Branch prediction perspectives using machine learning

Veerle Desmet and Koen De Bosschere

Ghent University
Department of Electronics and Information Systems (ELIS)
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
E-mail:{vdesmet,kdb}@elis.rug.ac.be

Abstract

Feeding pipelines with a constant flow of useful instructions is of primary importance for the overall processor performance. This constant feeding process is especially interrupted by conditional branches as the outcome of an instruction is typically known a few cycles after fetching. Modern microarchitectures overcome this difficulty by using a branch predictor, which predicts the direction of the branch in the fetch unit. As one instruction out of eight represents a branch and a misprediction seriously affects the performance, accurate branch prediction is paramount. In this paper, we discuss some perspectives that machine learning techniques can offer for further improving the accuracy of branch predictors.

1 Introduction

Deeply pipelined computer architectures as we know them today rely on a fetching mechanism that provides one or more useful instructions every clock cycle. This constant feeding process encounters difficulties because of the control dependencies, which determine the flow of the program at run time. Especially conditional branches give a problem: the next instruction to be executed is not known until the branch condition (e.g. argument equals zero) is computed. This computation typically completes 3-14 cycles after the branch has entered the pipeline, meanwhile no further instructions can be fetched.
To solve this situation, an essential part of modern microarchitectures consists of branch prediction. 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 method enables a constant pressure on the pipeline but involves additional complications for handling speculative instructions and verifying the prediction. Of course the branch instruction itself is 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 executes. This means that on a misprediction there is even an extra penalty compared to not predicting. As pipelines deepen and the number of instructions issued per cycle increases, the penalty for a misprediction also increases. Current branch predictors already reach 95% prediction accuracy and continuously improvements and 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 [7].
The next section gives an overview of the classical branch prediction schemes that are currently used in modern processors.

2 Prediction schemes

Branch prediction distinguishes two classes of prediction schemes: static and dynamic. Static branch prediction associates a fixed prediction for each static branch at compile-time, while dynamic prediction makes a prediction at run

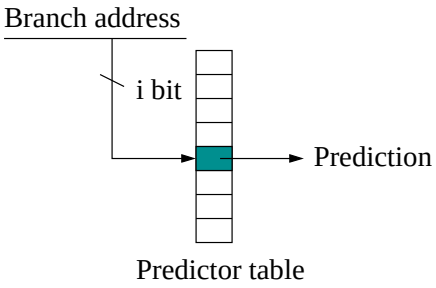

Figure 1: Bimodal predictor

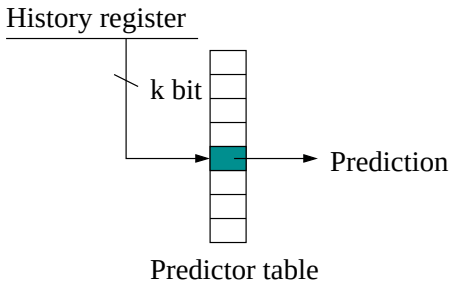

Figure 2: Global predictor

time. As branches tend to be mostly taken, the static strategy 'predict always taken' achieves prediction accuracies up to 65% [8]. Improved static predictors use profile information and program-based heuristics [1] to get accuracies up to 75%. More recent research on static prediction in [2] does not step over 80% accuracy, which is worse than almost any dynamic scheme. Therefore we only focus on dynamic branch prediction and we start with an overview of classical prediction schemes: bimodal, local, global, gshare, two-level and hybrid branch predictors.

2.1 Bimodal

The bimodal prediction scheme [8] in Figure 1 is very simple: a prediction table is indexed by the program counter or branch address and the contents is used to generate the prediction. Each table entry keeps a 2-bit saturating counter. If the counter value is above a certain threshold, e.g. 2, the branch is predicted taken and if it is not the prediction is not-taken. The idea behind this indexing scheme is that all instances of a static branch use the same entry and this performs well because previous outcomes provide useful information to predict the next occurrence. When the branch resolves, the effective direction is known and the used counter is incremented or decremented on a taken respective not-taken branch.

2.2 Global

A global predictor uses the *global history*, i.e. the outcomes of recently resolved branches, for indexing the counter table. The global history is stored

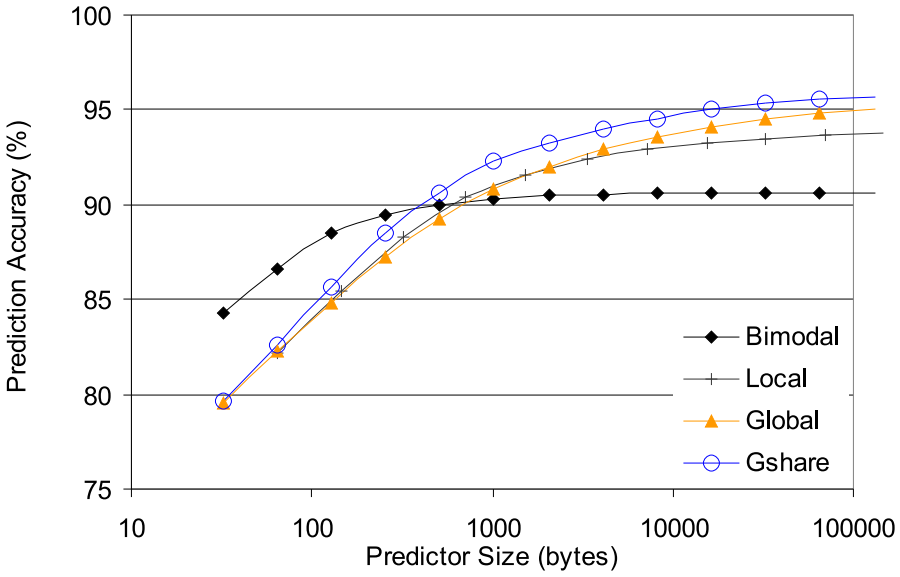

Figure 3: Prediction accuracy versus predictor size

in a register in which the outcome of a resolved branch is shifted during update. This scheme is able to take advantage of other recent branches to make a prediction and it performs better for sufficiently large predictors. This is illustrated in Figure 3, where the prediction accuracy is plot for varying predictor size. The most important predictor sizes lay between 1000 and 10000 bytes: smaller predictors are not accurate enough and larger predictors require too much hardware for little improvement.

2.3 Local

Figure 4 illustrates the local prediction scheme where two prediction tables are managed. As in the bimodal predictor the first table is indexed by the program counter, but now the exact *local history* is kept in the table. The local history indexes the second table where a saturating counter value indicates the branch direction prediction. The extra level allows that different local histories by a particular static branch use different saturating counters and therefore the prediction can be made more accurate. For example a loop as *for (i=0; i < 4; i++)* $\{\ldots\}$ corresponds with a branch that executes the

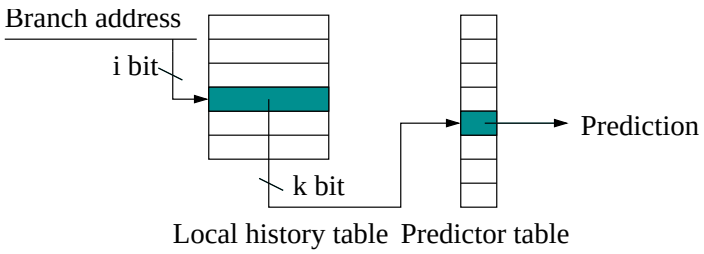

Figure 4: Local predictor

pattern *taken–taken–taken–not taken*. When the loop is executed a number of times, the next outcome follow from the previous three direction and a local predictor makes correct predictions. In Figure 3 a local predictor is considered where both prediction tables have the same number of entries, or $i = k$. When the branch resolves, counter value and local history are updated.

2.4 Gshare

Another indexing strategy is used in the gshare predictor proposed by McFarling [6]. The scheme in Figure 5 tries to combine good points of both bimodal and global prediction by using the XOR-result of branch address and global history as index in the prediction table. This XOR-operation gives more information than either component alone. Especially for large table sizes the gshare predictor performs very well with accuracies up to 95% as shown in Figure 3. A 8K-gshare predictor is used in the AMD K6 architecture and achieves a prediction accuracy of 95%.

2.5 Two-level adaptive prediction

Yeh and Patt [9] proposed the two-level adaptive prediction scheme that is shown in Figure 6. All the above-discussed predictors can be seen as special cases of this general structure. The branch address indexes the first level and localises a branch history register. The second level table is organised as $2^j \times 2^k$ elements with prediction information, e.g. 2-bit counters. While the branch history selects the row, the option for the column is chosen by the branch address. During update the used table information is updated with

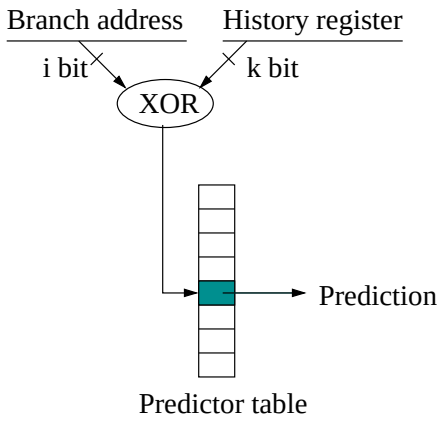

Figure 5: Gshare predictor

the most recent branch outcome. Athlon is a recent architecture using some variant of this two-level prediction scheme: 95% prediction accuracy with a 4K predictor.

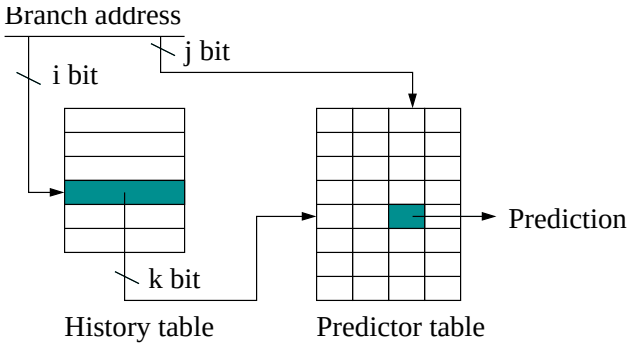

Figure 6: General structure of a two-level adaptive predictor

2.6 Hybrid predictors

Each of the presented prediction schemes has specific advantages and disadvantages. A natural question is whether they can be combined to make a new predictor with better prediction accuracy. In a hybrid predictor, two different predictors generate a prediction while a third predictor serves to

select which predictor to use. Hybrid predictors achieve higher prediction accuracies, but the required hardware is over double the size of a simple predictor. In the Alpha 21264 microprocessor [5] the branch predictor is a hybrid prediction scheme that dynamically chooses between local and global history to predict the direction of a given branch.

3 Machine learning

Two areas in which machine learning techniques already have largely contributed are image and speech recognition. Indeed, classification and function approximation are both examples in which machine learning techniques can achieve good results. We can see branch prediction as either of these situations. First, branch prediction can be considered as a classification problem as it classifies each of the dynamically executed branches in predicted taken or predicted not-taken. Also the learning concept is present: when the branch resolves the correct branch direction is available. Secondly, for a given set of history information there exists a function that perfectly predicts the branch outcomes. It is this program dependent function that all branch predictors strive to learn. A possible option is learning the function for each static branch separately and this is in fact what classical predictors do when indexing on the basis of the branch address.
Common techniques for handling classification and function approximation include the use of neural networks. Neural networks are mathematical constructs that are often thought to model biological nervous systems. They are composed of a set of nodes that are highly interconnected, mostly represented in a layer structure in which each interconnection has assigned a weight. In the rest of this paper we will focus on perspectives with neural networks but first we discuss the perceptron predictor.

3.1 Perceptron predictor

Recently, Jiménez *et al.* introduced a perceptron branch predictor [4] that successfully used a single layer perceptron in the branch prediction domain. The working of the perceptron predictor is illustrated in Figure 7. It is essentially a two-level prediction mechanism but uses perceptrons in stead of the classical saturating counters. In the perceptron all nodes in the input

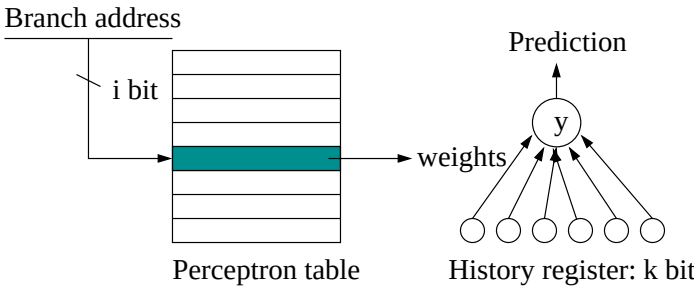

Figure 7: Perceptron predictor

layer represent a bit of the global history while the weights vector is taken from the perceptron table. The perceptron output y is computed as the dot product of weights and input vector and serves to decide whether the branch is predicted taken or not-taken. During update a weight is incremented when the branch outcome agrees with the corresponding input bit, and is decremented when it disagrees. The stronger the correlation, i.e. agreement or disagreement, the larger the weight and the higher it contributes to the output and final prediction. The weights are only updated on a misprediction or when y is a value close to zero indicating a poorly convinced prediction. A single layer perceptron is easy to understand but has a limitation to perfect learning so-called linearly separable functions. Nonetheless Figure 8 shows that the perceptron predictor outperforms classical schemes with accuracies up to 96%. The main reason for the good results is the ability to make use of longer history lengths [4]. Classical schemes that use the history as an index into a table require space exponential in the history length, while the perceptron predictor requires space linear in the history length. For example, a 4K-perceptron predictor works best with a history length of 24. At the same hardware budget gshare uses the maximal possible history length of 14.

4 Perspectives

It is clear that the perceptron predictor generates good predictions in an alternative way and many more options are possible with neural networks. Basically they include adding an extra layer to the network to form a multi-layer perceptron and changing input types such as adding local history in-

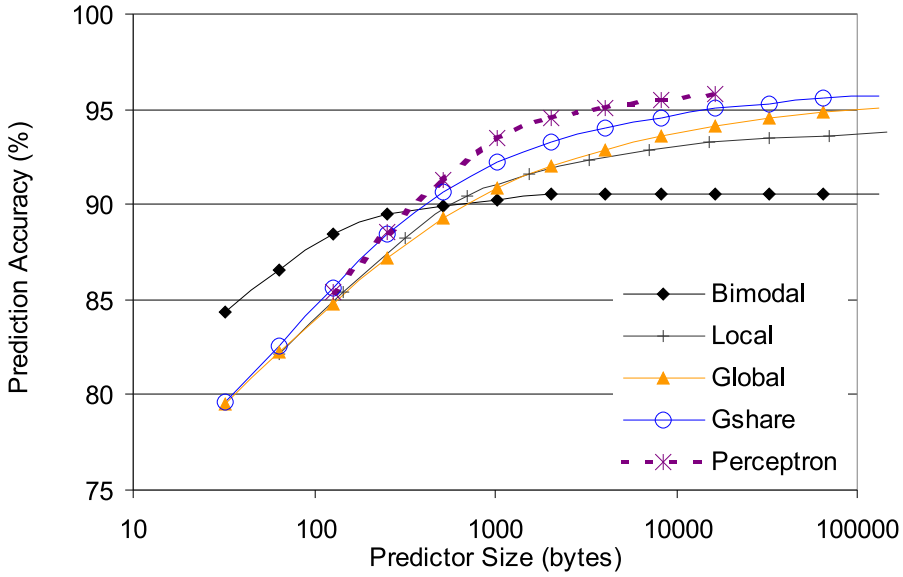

Figure 8: Accuracy of the perceptron predictor

formation. During the design phase one should determine the representation of the weights: in case of the perceptron predictor signed integers are used. Another key decision is the choice of the learning algorithm, which we will discuss together with some important properties in the next subsections.

4.1 Ignoring irrelevant information

An attractive property of neural networks is their capacity to ignore irrelevant information. Indeed, by assigning small weights to useless information it is possible to neglect the corresponding inputs. Moreover, a simple analysis of the weights can determine irrelevant inputs as well as the most relevant information. In this way we can search the most relevant information to improve the prediction schemes of today.

4.2 Learning algorithm

The most intelligent part of machine learning techniques is the learning algorithm, which in general consists of two parts: when to learn and how to

learn. The learning algorithm can vary from easy incrementing/decrementing weights like proposed in the perceptron predictor to more complex adaptation of weights to fulfill a certain error criteria. As mentioned in [4], the learning algorithm must be efficiently implementable in a branch predictor. Therefore simply incrementing/decrementing is preferable as the adaptation can be done in parallel. The second point in the learning algorithm is the condition on which further learning is forced. The condition is strongly recommended because it avoids the problem of overtraining occurring when the network needs a long time to adapt on new situations and which results in a serious performance loss. Possible conditions for the learning algorithm include:

- Only learn after a misprediction
- Only learn when the output value does not reach a certain threshold or after a misprediction, i.e. this strategy that is used in the perceptron predictor.
- Once a certain level of accuracy is obtained stop the learning; restart when the accuracy is decreasing
- Keep on training

Variants can use different learning strategies upon a correct prediction or misprediction. We conclude that many options can be evaluated in order to find the best strategy for branch prediction.

4.3 Confidence

Neural networks provide an additional advantage: a confidence-level. This is because the network output is not a Boolean value, but a number proportional to the certainty that the branch is taken. Some classical branch predictors explicitly compute a confidence [3], but here the confidence comes for free.

4.4 Grey box strategy

So far we present machine learning as a technique in which predictions follow automatically providing good predictive capabilities. Because the strategy

in neural networks can only partly be understood by human beings, they are often called black boxes. On the other hand, we must admit that the classical prediction schemes in Section 2 also perform well and rely on easy understandable and clear ideas. In this way they are considered as white boxes. A last perspective is the idea to provide some white box features as hints to machine learning techniques in order to improve them. For example we can try to help the system to adapt faster in some cases.

5 Conclusion

In this paper we explained why microarchitectures need branch prediction schemes. An overview of classical prediction schemes was given and we concluded that improvement above 95% is necessary to allow higher IPC values in future architectures. Machine learning techniques appear to give alternative prediction possibilities and their design shows many options. A first and very simple machine learning technique was used in the perceptron predictor, which achieves more than 96% prediction accuracy. General perspectives for future improvements of branch predictors include relevant information selection, identifying good learning rule and exploiting grey box alternatives.

6 Acknowledgements

Veerle Desmet is supported by a grant from the Flemish Institute for the Promotion of the Scientific-Technological Research in the Industry (IWT).

References

[1] Thomas Ball and James R. Larus. Branch prediction for free. *ACM SIGPLAN Notices*, 28(6):300–313, 1993.

[2] Brad Calder, Dirk Grunwald, Michael Jones, Donald Lindsay, James Martin, Michael Mozer, and Benjamin Zorn. Evidence-based static branch prediction using machine learning. *ACM Transactions on Programming Languages and Systems*, 19(1):188–222, January 1997.

[3] Erik Jacobsen, Eric Rotenberg, and James E. Smith. Assigning confidence to conditional branch predictions. In *International Symposium on Microarchitecture*, pages 142–152, December 1996.

[4] Daniel A. Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture*, pages 197–206, January 2001.

[5] R.E. Kessler, E.J. McLellan, and D.A. Webb. The alpha 21264 microprocessor architecture. In *In Proceedings of International Conference on Computer Design*, pages 90–105, October 1998.

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

[7] Henk Neefs, Koen De Bosschere, and Jan Van Campenhout. An analytical model for performance estimation of modern data-flow style scheduling microprocessors. In *Proceedings of the 22nd Euromicro Conference: Short Contributions*, pages 2–7, 1996.

[8] James E. Smith. A study of branch prediction strategies. In *Proceedings of the Eighth International Symposium on Computer Architecture*, pages 135–148, May 1981.

[9] Tse-Yu Yeh and Yale N. Patt. Two-level adaptive training branch prediction. In *Proceedings of the 24th Annual International Symposium on Microarchitecture*, pages 51–61, November 1991.