Improved Static Branch Prediction for Weak Dynamic Predictions

Veerle Desmet, Lieven Eeckhout, Koen De Bosschere

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

Abstract

Branch prediction is the process of predicting the outcome of conditional branches before they are actually executed. Branch prediction techniques are divided into two broad classes: static and dynamic. Static branch prediction associates a fixed prediction to each static branch at compile time whereas dynamic strategies makes a prediction at run time. Today's most accurate static method is evidence-based static prediction (ESP), which uses machine learning to generate a prediction based on a set of static features [3]. By extending the static feature set of ESP, we further reduce the miss rate of 17.4% for ESP towards 11.6% (SPEC 2000). Although almost all dynamic predictors do better than static methods, highly accurate static prediction can serve as prediction for weak dynamic predictions in e.g., gshare.

1 Introduction

Deeply pipelined computer architectures as we know them today rely on a fetch 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 cause difficulties: 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.

After a brief related work, this paper starts with the discussion of an improved static branch prediction scheme. Then we explore where highly accurate static branch prediction can serve as prediction for weak dynamic predictions.

2 Related work

Ball and Larus presented in [1] a set of heuristics for static branch prediction by encoding knowledge about common programming idioms. They use information about branch opcode, operands and characteristics about the branch successor blocks.

Calder *et al.* use neural networks and deci-

sion trees to generate a general profile to statically predict the branch behaviour in new unseen programs [3]. Their so-called evidence-based static prediction method is, to our knowlegde, the best static prediction scheme.

Important contributions in dynamic branch prediction include the proposal of the *gshare* predictor in [4], which makes a prediction based on the XOR-combination of branch address and global history. Gshare is considered as one of the best dynamic predictors known today (except for hybrid predictors).

The idea of combining static and dynamic branch prediction is recently proposed in [5]. Based on profiling analysis they select branches for static prediction. When an appropriate set of statically predicted branches is chosen, it reduces the effect of aliasing in the dynamic predictor and results in higher accuracies.

3 Improved static prediction

Calder *et al.* [3] propose a technique which they call evidence-based static prediction (ESP). The main idea of their work is to use a set of static features associated with each branch as input to some machine learning technique. The ESP feature set contains information about branch opcode, branch direction, branch operands, whether the basic block is a loop header or not and also characteristics about dominator, postdominator and ending type of both successor basic blocks; a total of 30 static features. On average, ESP branch prediction results in a miss rate of 17.4%. [1]

For the experimental results, we use the SimpleScalar/Alpha simulator (sim-bpred) [2]. We simulate the 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`. We instrumented each program, i.e. we determine for every conditional branch instruction the appropriate values of the static features. Besides this, we run the programs for completion with different inputs and measure the execution frequency and branch probability. Based on this information we label the executed branches with their most chosen direction.

[1] Calder reported for the SPEC92 benchmark suite and other programs 20% compared to 25% by heuristics.

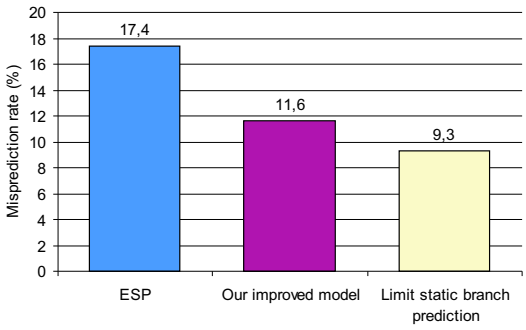

Figure 1: Miss rate for profile of SPEC 2000

The decision trees are built with C4.5 [6], a widely used tool for constructing decision trees and also used in [3]. C4.5 constructs a decision tree with respect to minimize the static prediction error and therefore it does not take into account the execution frequency of the branches. Of course, we consider this number of dynamic executions in the reported prediction accuracies, 17.4% for ESP.

We extent the model of ESP by adding some more features that are available at compile time. These additional features include looplevel, basic block size, register used in branch instruction and dependency distance (in number of instructions) between register producing instruction and actual branch. Again, we build a decision tree model for data based on the extended features set and measure the miss rate for a general profile of SPEC 2000 programs. The result is shown in Figure 1. Our improved model has a miss rate of 11.6%compared to 9.3% miss rate of the perfect static branch predictor. Although the extended model uses more features and therefore makes it easier to diversify between different branch structures, this new prediction model will not automatically achieve higher prediction accuracies. Remember that the tree model is built for assigning the mostly taken direction to each branch and not for optimizing the dynamic prediction accuracy.

Although existing dynamic branch prediction schemes achieve lower miss rates (down to 5%) than even the perfect static branch predictor, these accurate static predictions can serve as alternative for some dynamic predictions, resulting in less mispredictions.

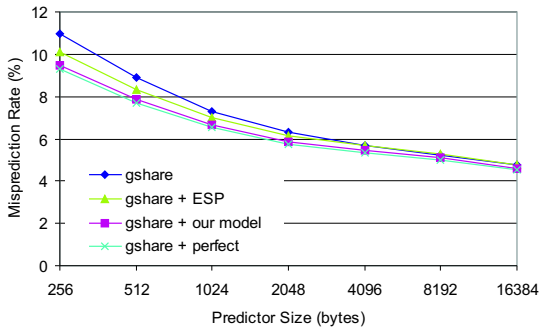

Figure 2: Gshare predictor with static prediction for weak states

4 ... for weak dynamic predictions

A gshare predictor bases its prediction on the state of a (mostly) 2-bit saturating counter. For this branch predictor, we observe that predictions corresponding with a counter value that is in a *weak* state (e.g., 1 and 2 for a 2-bit counter), the branch behaviour closer reflects the perfect static prediction than its dynamic prediction. As thus we measure the effect of replacing these weak dynamic predictions by a static prediction.

The results are shown in Figure 2 for different gshare predictor sizes. Beside the original gshare, three extensions are shown according to the different static prediction possibilities discussed in the previous section. We see that our model is very close to the perfect static prediction and that for a 256-byte gshare with our combination the miss rate is reduced to 9.4% compared to 10% for ESP. The effect of replacing weak predictions by static predictions seriously diminishes as larger predictor sizes are considered.

5 Conclusion

By extending the static feature set of ESP, we further reduce the static branch prediction miss rate of 17.4% for ESP towards 11.6% (SPEC 2000). This is close to the limit for perfect static branch prediction that reaches a miss rate of 9.3%. These highly accurate static predictions can serve as prediction for weak dynamic predictions. For small predictor sizes this results in significant lower miss rates.

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

[2] D. Burger, T. M. Austin, and S. Bennett. Evaluating future microprocessors: The SimpleScalar Tool Set. Technical report, Computer Sciences Department, University of Wisconsin-Madison, July 1996.

[3] B. Calder, D. Grunwald, M. Jones, D. Lindsay, J. Martin, M. Mozer, and B. Zorn. Evidence-based static branch prediction using machine learning. *ACM Transactions on Programming Languages and Systems*, 19(1):188–222, Jan. 1997.

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

[5] H. Patil and J. Emer. Combining static and dynamic branch prediction to reduce destructive aliasing. In *Proceedings of the 6th International Symposium on High Performance Computer Architecture*, pages 251–262, Jan. 2000.

[6] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.