THE EASE BRANCH PREDICTOR

Serene Banerjee, Lizy K. John, and Brian L. Evans

Dept. of Electrical and Computer Engineering
The University of Texas at Austin, Austin, TX 78712-1084 USA
{serene,ljohn,bevans}@ece.utexas.edu

ABSTRACT

Wide issue processors with many pipeline stages require efficient branch prediction for high throughput. In this paper, we propose an Embedded, Architecturally Simple and Efficient (EASE) branch predictor, which performs well for programs having a small number (100) and a large number (16,000) of instructions. EASE uses a combination of a G-share predictor and a one-level predictor. In EASE, one-level predictor is active until the G-share predictor warms up. For an example program of 100 instructions, the one-level, two-level, G-share, and EASE predictors give misprediction percentages of 36%, 38%, 41%, and 36%, respectively. The corresponding figures for an example program with 16,000 instructions are 9.5%, 5.6%, 3.3% and 3.3%, respectively. For the gcc benchmark, the EASE predictor outperforms hybrid and cascaded predictors by giving a misprediction percentage of 7.89%. When compared to hybrid and cascaded predictors, the EASE predictor has fewer counters and no decision circuitry, which makes it more efficient to implement in VLSI.

Keywords: branch prediction, hybrid predictor, cascaded predictor, superscalar processor

1. INTRODUCTION

Trends in the design of out-of-order superscalar processors has been to increase the issue width and pipeline depth. In the mid 1990s, superscalar processors were generally two-way (such as HP PA 7200) or four-way (such as HP PA 8200 and UltraSparc), and had 5 or 6 pipeline stages. Today, these figures have exploded to 40/64-way superscalar having 10–12 pipeline stages, such as Intel's Pentium III, Intel's IA64 Itanium, AMD's Athlon and IBM/Motorola's PowerPC G4 processors. Pipeline flushes due to branch mispredictions is a serious hindrance to achieving high performance.

The better methods for branch and target prediction include correlation-based predictors [1, 2], counter-based predictors [3], the G-share predictor [4], and hybrid predictors [5, 6]. Counter-based methods outperform correlation-based methods when the buffer size is small [3]. As the buffer size increases, the improvement of the correlation-based methods over the counter-based methods increases. The G-share predictor incorporates randomization to improve prediction accuracy. It applies a bitwise exclusive OR to the global branch history and branch address to generate a new random address. Hybrid predictors switch between two methods based on contextual information.

In this paper, we embed a counter-based one-level predictor into the G-share predictor to create an Embedded, Architecturally Simple and Efficient (EASE) branch predictor. EASE is not a hybrid predictor. EASE outperforms a hybrid predictor [5] and a cascaded predictor [7], and yields comparable results to a recent hybrid predictor [8], for the SPEC95 benchmarks [9, 10]. The EASE simulator is available at

http://www.ece.utexas.edu/~serene/software/ease/

Section 2 reviews different types of branch predictors. Section 3 describes the proposed EASE branch predictor. Section 4 compares EASE with one-level, two-level, and G-share branch predictors. Section 5 concludes the paper.

2. BACKGROUND

Many counter-based one-level predictors use a table consisting of two-bit counters. The table is indexed with the address of the instruction being executed. Depending on the counter value, which corresponds to the address, the branch is either predicted to be taken or not taken. If the branch is taken, then we predict its target address from the branch target buffer. The percentages in this paper refer to the target misprediction percentages.

Correlation-based two-level predictors use a global counter to track the history of branch predictions. The

The authors of this paper are supported in part by the grant CCR9796098.

table of counters is indexed with both the branch prediction history and the address bits of the instructions being executed. The G-share predictor is inherently a two-level predictor, because it considers history of the recent branches. Instead of maintaining a two-dimensional array of counters, however, it uses a one-dimensional array of counters indexed using the exclusive OR of the PC with the recent branch patterns.

3. PROPOSED PREDICTOR

When program size is small, i.e. the number of instructions executed is small, the counter-based one-level predictor has better prediction than the G-share predictor. Because of the inherent random nature of the G-share predictor, it takes more time to warm up, and thus suffers from more cold start misses. For small programs, the program execution is completed by the time that the G-share warms up.

In the EASE predictor, we embed a small one-level branch predictor in the G-share predictor. That is, if the G-share predictor is of size 4K entries, then we are using a one-level predictor of size 1K entries. When the processor starts executing a program, it starts with the one-level predictor. We also define "cutoff" as the number of instructions after which the one-level predictor is abolished and prediction is performed with the G-share predictor. The cutoff is chosen by experimental means. It was found that if the cutoff were equal to the number of entries of the one-level branch predictor, then good prediction accuracies were obtained for the SPEC95 benchmarks. However, for some individual benchmarks prediction accuracies could be increased by changing the cutoff. Unlike hybrid predictors, once the G-share is chosen, it does not go back into the one-level stage, unless the CPU starts executing a new program altogether.

Given the size of the G-share predictor, we find a smaller one-level predictor, which when embedded into the G-share predictor, improves the prediction accuracy. The performance is measured over a generic set of small and large programs. The sizes for the G-share predictor and the corresponding one-level predictor are summarized in Table 1.

The input file to the simulated EASE predictor consists of the PC address, the target address and the information whether the branch is actually taken or not. Depending on the PC address, the decision whether the branch is taken or not is read from the branch history buffer. If the branch is predicted taken and is actually taken, then we obtain the target prediction from the branch target buffer. Thus, again depending on the PC address, the branch target buffer is read. If the predicted target address matches the actual branch address, then the prediction is correct. Also, if the branch is predicted not taken and is actually not taken, we do not read the branch target buffer and say that target prediction was correct. We increment the target prediction accuracy counter for either of the above cases. The number of mispredictions is then obtained by subtracting the total number of correct target predictions from the total number of branch instructions executed.

Size of G-share predictor	Size of one-level predictor
64 entries	16 entries
128 entries	16 entries
256 entries	16 entries
512 entries	128 entries
1024 entries	256 entries
2048 entries	512 entries
4096 entries	512 entries
8192 entries	1024 entries
16384 entries	1024 entries

Table 1: The size of one-level predictor, embedded in EASE for different sizes of G-share predictors.

4. EXPERIMENTAL RESULTS

The EASE branch predictor simulator was written in the C language. At first synthetic programs were used to validate the predictor. For the synthetic programs specific repetitive patterns were assumed for the taken/not taken information. This pattern was iterated 100 and 16,000 times, respectively, to generate a relatively smaller and a larger test program. For the large program, the G-share predictor gives better prediction accuracy, whereas for the small program, the one-level predictor gives better prediction accuracy. This is because the G-share predictor suffers more cold start misses than the one-level predictor of the same size.

The EASE predictor utilizes both the randomization capability of the G-share predictor as well as the quick warm-up time of the one-level predictor. So, for both cases, the EASE predictor gives better prediction accuracy than the G-share, one-level, and two-level predictors. The results for one-level, two-level, G-share and the EASE predictors for these synthetic programs are summarized in Figs. 1 and 2. For both the small and the large synthetic programs, the EASE predictor does better.

We interfaced the EASE branch predictor with Shade [11], which is a tracer for SPARC applications, to test the predictor on the SPEC95 benchmarks. Fig. 3 compares the results for gcc with those for a hybrid predictor [5] and a cascaded predictor [7]. For the gcc benchmark, the EASE predictor gives better prediction results.

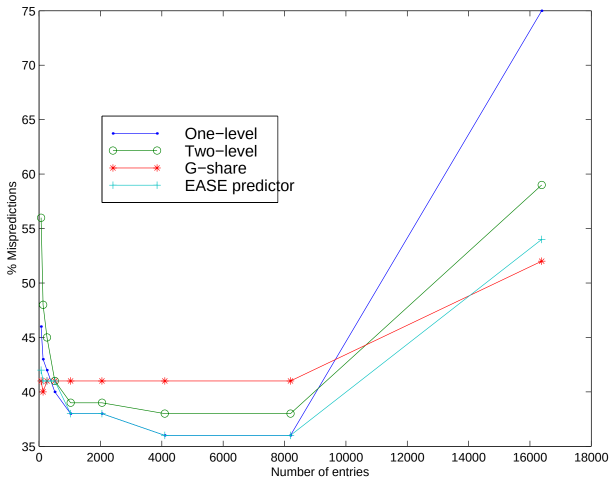

Figure 1: Misprediction percentages for one-level, two-level, G-share, and EASE predictors for a sample 100 instruction program.

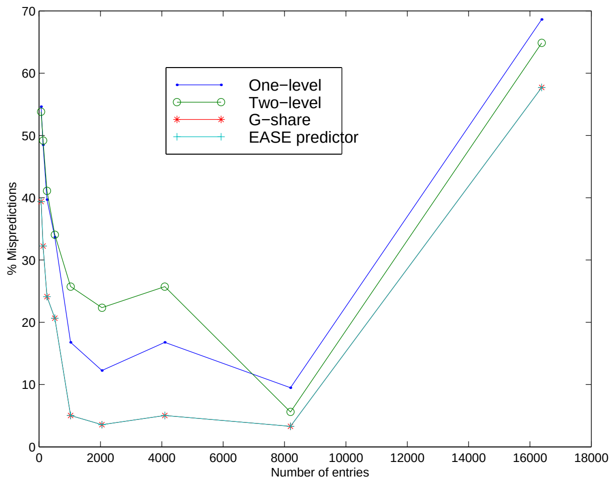

Figure 2: Misprediction percentages for one-level, two-level, G-share, and EASE predictors for a sample 16,000 instruction program.

For the other SPEC95 benchmarks— go, compress, ijpeg, li, m88ksim, and perl— the misprediction percentages obtained with Shade are given by Table 2 and Fig. 4. The EASE predictor performs comparably for all of these benchmarks except for go. Hybrid predictors on average have misprediction percentages of 1-14% for the SPEC95 benchmarks [9, 10]. The EASE branch predictor, on the other hand has misprediction percentages of 1.8-18% for the SPEC95 benchmarks. The EASE predictor yields comparable results for some of the benchmarks, although it does not do well for the go benchmark.

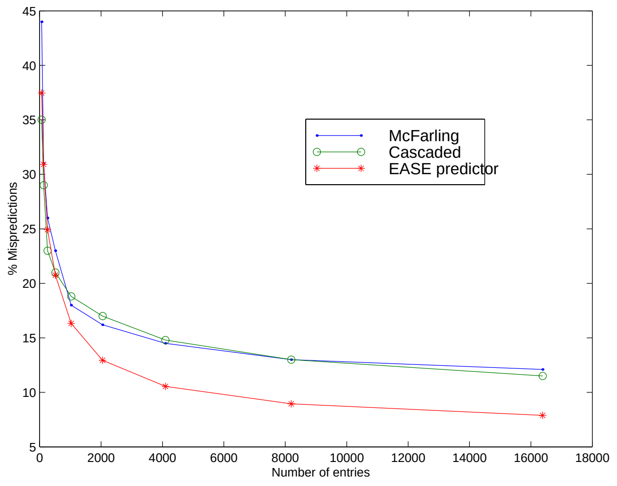

Figure 3: Misprediction percentages for McFarling's hybrid predictor, the cascaded predictor, and the EASE predictor for the gcc benchmark.

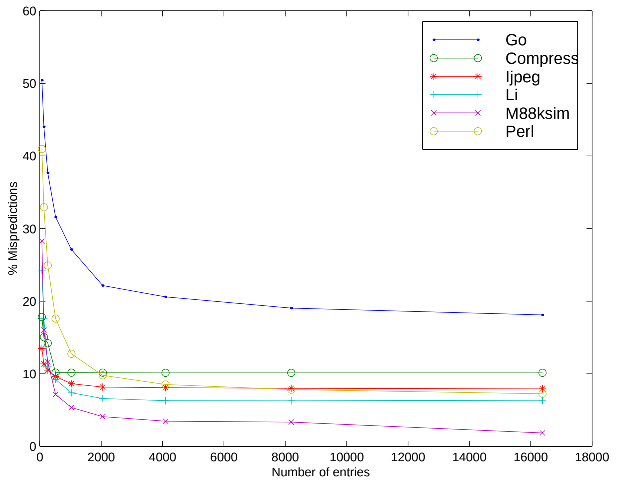

Figure 4: Misprediction for the EASE predictor for the SPEC95 benchmarks.

5. CONCLUSION

The EASE branch predictor is comparable in terms of prediction accuracy but has much simpler architecture than McFarling's hybrid predictor. The EASE predictor requires many fewer counters (as our one-level predictor is smaller in size compared to the G-share

G-share size	go	compress	gcc	ijpeg	li	m88ksim	perl
64 entries	50.44	17.83	37.46	13.47	24.27	28.26	40.95
128 entries	44.03	15.03	30.92	11.36	17.64	16.02	32.93
256 entries	37.67	14.22	24.93	10.49	10.97	11.62	24.93
512 entries	31.59	10.16	20.75	9.63	9.20	7.16	17.59
1024 entries	27.13	10.16	16.32	8.64	7.38	5.35	12.75
2048 entries	22.16	10.14	12.94	8.16	6.59	4.09	9.78
4096 entries	20.59	10.13	10.55	8.08	6.39	3.47	8.52
8192 entries	19.04	10.12	8.95	7.99	6.37	3.34	7.82
16384 entries	18.11	10.12	7.89	7.93	6.35	1.84	7.24

Table 2: Misprediction percentages of various benchmarks, with varying size of the G-share predictor, in the EASE branch predictor. Simulations were done using SimpleScalar, Version 2.0.

predictor). The EASE predictor requires no additional hardware to make decisions for context switching. The elimination of context switching also eliminates the delay involved in the decision process.

The EASE predictor outperforms the cascaded predictor in terms of prediction accuracy and architectural simplicity. In processors where both speed of execution and architectural simplicity are crucial, the EASE predictor serves as a better alternative. The EASE predictor is also well suited for general-purpose processors which not only execute larger programs, but at times have to execute smaller programs as well.

6. REFERENCES

[1] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt, “An Analysis of Correlation and Predictability: What Makes Two-level Branch Predictors Work,” in *Proc. ACM/IEEE Int. Sym. on Microarchitecture*, vol. 1, pp. 52–61, June 1998.

[2] P.-Y. Chang, H. Hao, and Y. N. Patt, “Target Prediction for Indirect Jumps,” in *Proc. ACM/IEEE Int. Sym. on Microarchitecture*, vol. 1, pp. 274–283, June 1997.

[3] B. Fagin and A. Mital, “The Performance of Counter- and Correlation-Based Schemes for Branch Target Buffers,” *IEEE Trans. on Computers*, vol. 44, pp. 1383–1393, Dec. 1995.

[4] S. Onder, X. Jun, and R. Gupta, “Caching and Predicting Branch Sequences for Improved Fetch Effectiveness,” in *Proc. Int. Conf. on Parallel Arch. and Compilation Tech.*, vol. 1, pp. 294–302, Oct. 1999.

[5] S. McFarling, “Combining Branch Predictors,” in *WRL Technical Note, Digital Equipment Corporation*, vol. TM-36, June 1993.

[6] D. Grunwald, D. Lindsay, and B. Zorn, “Static Method in Hybrid Branch Prediction,” in *Proc. Int. Conf. on Parallel Arch. and Compilation Tech.*, vol. 1, pp. 222–229, Oct. 1998.

[7] K. Driesen and U. Holzle, “The Cascaded Predictor: Economical and Adaptive Branch Target Prediction,” in *Proc. ACM/IEEE Int. Sym. on Microarchitecture*, vol. 1, pp. 249–258, Nov. 1998.

[8] M. Evers, P.-Y. Chang, and Y. N. Patt, “Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches,” in *Proc. ACM/IEEE Int. Sym. on Microarchitecture*, vol. 1, May 1996.

[9] S. Manne, A. Klauser, and D. Grunwald, “Branch Prediction Using Selective Branch Inversion,” in *Proc. Int. Conf. on Parallel Arch. and Compilation Tech.*, vol. 1, pp. 48–56, Oct. 1999.

[10] H. Patil and J. Emer, “Combining Static and Dynamic Branch Prediction to Reduce Destructive Aliasing,” in *Proc. Int. Sym. on High-Performance Computer Arch.*, vol. 1, pp. 251–256, Jan. 2000.

[11] T. Austin and D. Burger, “The SimpleScalar Tool Set, Version 2.0,” in *Technical Report, University of Wisconsin-Madison Computer Sciences Department*, vol. TR-1342, June 1997.