Proceedings of Machine Learning Research vol 144:1–5, 2021

CSE240A Branch Predictor Project

Quan Luo QULUO@UCSD.EDU

Abstract

To realize the performance potential of today's widely-issued, deeply pipelined superscalar processors, a good branch prediction mechanism is essential. In this project, we explore 2 kinds of branch predictor namely Gshare & Tournament branch predictor. We also implemented a custom predictor and compared their performance on some given benchmarks. Codes are open-sourced at CSE240A Branch Predictor Project.

1. Introduction

One of the key factors determining computer performance is the degree to which the implementation can take advantage of instruction-level parallelism. Perhaps the most critical limit to this parallelism is the presence of conditional branches that determine which instructions need to be executed next. Though it may increases hardware complexity, the performance can be boost significantly with good branch predictors.

The branch performance problem can be divided into two subproblems. First, a prediction of the branch direction is needed. Second, for taken branches, the instructions from the branch target must be available for execution with minimal delay. One way to provide the target instructions quickly is to use a Branch Target Buffer, which is a special instruction cache designed to store the target instructions. This project focus on the first part, which is predicting branch directions.

Branch prediction strategies have been studied extensively. The main trends to improve branch predictors is to reduce aliasing between two indices that map the same entry in Pattern History Table(PHT) and combine patterns to get better performance. Lots of work done to improve predictor performance and decrease the aliasing such as McFarling (1993), Sprangle et al. (1997) and Lee et al. (1997). In this project we mainly focus on G-share and Tournament branch predictor.

The rest of the project report is formulated as follow. This section will also introduce the mechanism of each predictor along with its advantages and disadvantages. We'll also introduce a custom branch predictor by ourselves. Section 2 will be focusing on the implementation of each branch predictor. Section 3 will be the observation about what changes the performance of these predictors and how the factors differ. In section 4 we give a brief interpretation of the observation and the final results conclusion of the project.

1.1. G-share BP

For global branch predictor, G-share by McFarling (1993) should be one of the most famous. Normally global branch predictor suffer from aliasing if we have only few bits for PHT. However, if there're enough address bits to identify the branch, we can expect the frequent history combinations to be rather sparse. To address the problem, G-share take advantage of that effect by hashing the branch address with the global history together. In particular, it can expect the exclusive OR of the branch address with the global history to have more information than either component alone.

Here comes the mechanism of G-share, just to use branch address bits to do XOR with the global history as the index in the counter table as shown in Figure 1. The advantage of G-share is obvious. By reducing the aliasing inside global PHT, the prediction will sure be better compared to methods without hashing. The disadvantage is that we haven't used the local history pattern of each branch, which definitely contains some information useful for predicting.

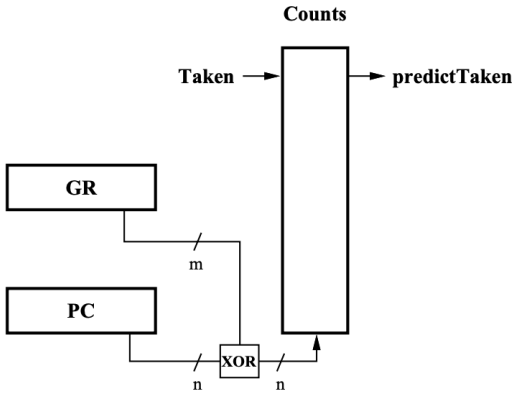

Figure 1: G-share Branch Predictor Mechanism

G-share evolves originally from Yeh and Patt (1992), a simple global branch predictor. As people found that aliasing is the biggest problem inside global PHT, a lot of mechanism developed to make the index sparse. One of the most famous one should be G-select by Pan et al. (1992), which the counter table is indexed with a concatenation of global history and branch address bits. As people found this is not enough for reducing aliasing, G-share comes out to use XOR instead of concatenating.

1.2. Tournament BP

To better combine local information and global information, Kessler et al. (1998) uses tournament branch predictor in the Alpha 21264 Micro-processor, which is our target branch predictor in our project.

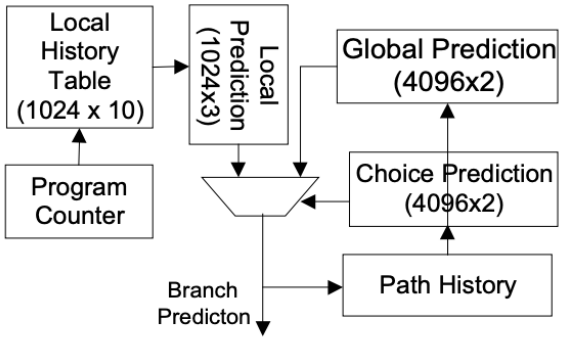

Figure 2: Tournament Branch Predictor Mechanism

As shown in Figure 2, the mechanism is also simple. Different from G-share, we also have a local predictor along with the global predictor. Apart from that, we have a choice predictor, which does not contain prediction of the branch but predict which predictor is better. By choice predictor, we either use the result of the local or global prediction.

Tournament branch predictor combines the advantage of local and global predictor and make full use of the information from local and global history. Disadvantage comes from the complexity and the essence of choice predictor. It's really hard to define which predictor is better only with the path history.

1.3. Custom BP

Our custom branch predictor adopt the idea of tournament predictor and moreover propose to combine it with modern technique such as predict by perceptron by Jimenez and Lin (2001). The idea is also simple. We keep the two predictors from tournament predictor as shwon in Figure 2 and add one more prediction by perceptron. At last we do a majority vote to decide which is the best as shown in Figure 3.

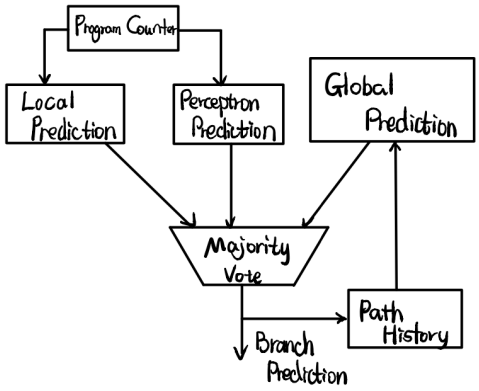

Figure 3: Custom Branch Predictor Mechanism

Our main idea is to combine the prediction of each mechanism that utilize different information. Perceptron needs time to train its weight and get better results while the other two converge faster. After the perceptron predictor gets trained, it serves as a better "choice predictor" as the one in tournament branch predictor. The majority vote can give us a better performance hopefully. The disadvantage comes from the complexity.

2. Implementation

In this section, we discuss the implementation of each branch predictors along with our research with respect to the specific predictor.

2.1. G-share & Tournament BP

For G-share branch predictor, according to discussion in Section 1.1, we use a uint_64t type variable to store the global history and a 2-bit counter table to store PHT. The 2-bit counter table is just a simple state machine with Strongly Taken(ST), Weakly Taken(WT), Weakly Not Taken(WN) and Strongly Not Taken(SN) to indicate current prediction. We use the XOR of PC address bits and current global history as the index. After each prediction we update the global history and PHT accordingly.

For tournament branch predictor, according to discussion in Section 1.2, we use one local predictor (which is also a 2-bit counter table) indexed by some PC address bits and another global 2-bit counter table predictor. The choice predictor is almost the same as the global predictor except that the content is Strongly Global(SG), Weakly Global(WG), etc. instead of ST/WT/WN/SN. It's only used to choose from the prediction of the mentioned local predictor and global predictor and does not propose branch prediction.

The training for global / local predictor is exactly the same as one in G-share. For choice predictor, we only update it when local / global predictor outputs different results. We move one

step towards the right one, i.e., if local predictor gets wrong while global predictor is correct, we move towards preferring the result of global predictor.

2.2. Custom BP

In our custom predictor, the implementation of the local / global predictor is exactly the same as in tournament predictor. The only part matters is the perceptron predictor which is stated in the paper by Jimenez and Lin (2001).

As shown in Figure 4, the prediction y is decided by some trainable weights w_i along with current history x_i. For each indexed PC branch address, we record a weight table initialized to 0. The final prediction y is given by

$$y = w_0 + \sum_{i=1}^{n} x_i w_i$$

where x_i is either -1 or 1 indicating NOT TAKEN and TAKEN respectively in the global history, w_i is the trainable weight and there's a w_0 bias term which is 1 in our setting.

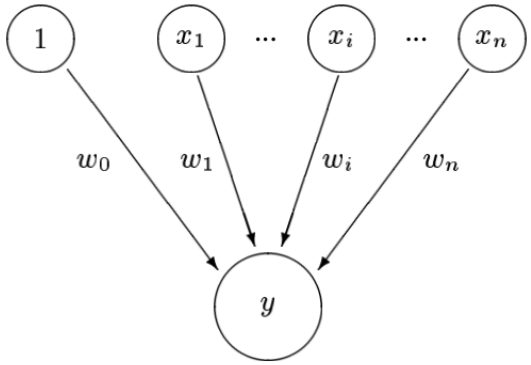

Figure 4: Perceptron Predictor

As for training section, we update the perceptron only if the prediction is incorrect. When it's incorrect we update the weights by $w_i := w_i + tx_i$ where t and x_i are either -1 or 1. t is the current outcome value. We also have a limitation of the maximum value and minimum value for these weights which is set to ± 10.

3. Observation

The results are shown in Table 1. The best branch predictor result is bold with red. Here the setting of G-share is 13 global history bits, setting for tournament predictor is 9 global history bits with 10 PC address bits and 10 local history bits. Setting for our custom predictor is 13 global history bits with 10 local history bits and 10 PC address bits.

	fp_1	fp_2	int_1	int_2	mm_1	mm_2
Static	12.128%	42.350%	44.136%	5.508%	50.353%	37.045%
G-share	**0.825%**	**1.678%**	13.839%	**0.42%**	6.696%	10.138%
Tournament	1.435%	6.221%	15.966%	0.733%	6.503%	9.942%
Custom	1.029%	2.542%	**13.632%**	**0.42%**	**4.585%**	**9.103%**

Table 1: Misprediction Rate in Dataset

Need to note that G-share generally did better in these tasks than tournament predictor. One possible reason is that aliasing made our tournament predictor suffer from heavy mistakes while G-share achieves sparse mapping.

As we could see, G-share still did better in some of the tasks but generally our custom predictor did the best. Here we can conclude that to combine all the useful information together might be the most significant things. We are using local / global history information along with a perceptron that deals with both and serve as a selector. We can also note that here the perceptron is obviously better than a 2-bit counter table choice predictor.

4. Results & Conclusion

In this project, we presented two existing methods for improving branch prediction performance and proposed one custom branch predictor. First we showed that using the bit-wise XOR of the global branch history and the branch address to access predictor counters gets us better performance. Next we show combining local and global 2-bit counter table branch predictor to utilize the local history table. Finally in order to boost the performance, we use a perceptron to serve as the choice predictor by doing majority vote in our custom design and outperform the two existing mechanism.

References

D.A. Jimenez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206, 2001. doi: 10.1109/HPCA.2001.903263.

R.E. Kessler, E.J. McLellan, and D.A. Webb. The alpha 21264 microprocessor architecture. In *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*, pages 90–95, 1998. doi: 10.1109/ICCD.1998.727028.

Chih-Chieh Lee, I.-C.K. Chen, and T.N. Mudge. The bi-mode branch predictor. In *Proceedings of 30th Annual International Symposium on Microarchitecture*, pages 4–13, 1997. doi: 10.1109/MICRO.1997.645792.

Scott McFarling. Combining branch predictors. Technical report, Citeseer, 1993.

Shien-Tai Pan, Kimming So, and Joseph T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS V, page 76–84, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 0897915348. doi: 10.1145/143365.143490. URL https://doi.org/10.1145/143365.143490.

Eric Sprangle, Robert S. Chappell, Mitch Alsup, and Yale N. Patt. The agree predictor: A mechanism for reducing negative branch history interference. *Conference Proceedings. The 24th Annual International Symposium on Computer Architecture*, pages 284–291, 1997.

Tse-Yu Yeh and Yale N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, ISCA '92, page 124–134, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 0897915097. doi: 10.1145/139669.139709. URL https://doi.org/10.1145/139669.139709.