Branch Predictor

Boyuan Tong
Computer Science
UC San Diego
btong@ucsd.edu

Abstract

In this project, I explore the use of three different branch predictors. Without branch prediction, the processor has to wait until the branch instruction has passed the execute stage before the next instruction can enter the fetch stage in the pipeline. Using branch prediction could improve the flow of the instruction pipeline. The branch predictors used are the G-share branch predictor, the tournament branch predictor, and a combined-Gshare-tournament branch predictor. The third custom branch predictor performs better than both the G-share and the tournament branch predictor in all 6 traces. Meanwhile, the maximum memory usage of the custom predictor does not exceed $(64K + 256)$ bits.

1 Introduction

Branch prediction is a method to improve the instruction pipeline's flow. Without branch prediction, the processor has to wait until the branch instruction has passed the execute stage before the next instruction can enter the fetch stage in the pipeline. The branch predictor attempts to avoid this waste of time by trying to guess which way a branch will go before executing it. If the branch predictor predicts correctly, it is expected to save a lot of time especially when branch instructions are frequent. However, the branch predictor also has a risk of wrong guesses. The wrong guesses, also known as branch misprediction, would waste more time as the pipelines are longer. Since modern microprocessors tend to have long pipelines, an advanced branch predictor is in high demand.

In this project, I implemented three different branch predictors and test their performance in six traces. The first branch predictor, the G-share branch predictor, uses only the program counter and the global history register. The second branch predictor, the tournament branch predictor, uses both local predictor and global predictor. A choice predictor is included to decide whose result to use. The third branch predictor, the combined-Gshare-tournament predictor, combines the benefit of the previous two branch predictors. By experimenting with these branch predictors in all six traces, the combined predictor outperforms both the G-share branch predictor and the tournament branch predictor. The details about all predictors and their implementation details are mentioned in the following sections. My code for these predictors is available here.

2 Branch predictors

In this section, I will introduce each of the three branch predictors. In each subsection, I would raise the reasons for using, implementation details, advantages, and disadvantages of the corresponding branch prediction.

2.1 G-Share Branch Predictor

The G-share branch predictor architecture is shown in Fig 1. The G-share branch predictor uses a pattern history table (PHT) whose indices are the XORs of the global history register and the lower bits of the branch's address. With ghistoryBits indicating the length of the global history kept, the number of entries in the PHT is $2^{ghistoryBits}$. Each PHT entry is a 2-bit predictor, which is a state machine with four states: strongly not taken, weakly not taken, weakly taken, and strongly taken. During the prediction process, the predictor XORs the program counter and the global history to select one entry from PHT and gets one of four states based on the entry value. During the prediction process, the real outcome will update the corresponding entry based on the state transitions and the global history register would be updated by shifting in new history to the least significant bit position.

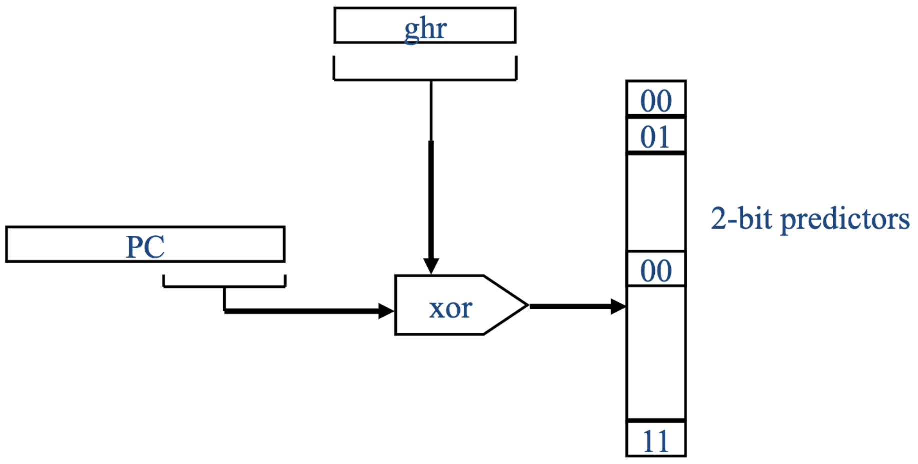

Figure 1: G-share branch predictor architecture. image credit: Branch Predictor[1]

By using global history, the predictor can capture any correlation between different branches in making the prediction. Compared with the G-select branch predictor, by XORing instead of concatenating the global history and branch address, the G-share branch predictor avoids the extremely frequent usage of some entries in the PHT if the branch addresses have the same lower bits.

However, since the G-share branch predictor only captures global history data, it cannot capture the patterns shown based on each branch address. Also, if different branches are uncorrelated, global history would be diluted by irrelevant information.

2.2 Tournament Branch Predictor

The tournament branch predictor architecture is shown in Fig 2. The G-share branch predictor is composed of a local predictor, a global predictor, and a choice predictor. The global predictor is similar to the G-share branch predictor except that it uses only global history instead of the XOR result as the PHT address indices. The local predictor has two tables: a branch history table (BHT) showing the local branch history of branch addresses with the same lower bits and a PHT showing the prediction based on the corresponding local branch history. The choice predictor is the same as the global predictor except that it predicts which predictor, global predictor or local predictor, to use instead of TAKEN or NOT TAKEN. During the prediction, the choice predictor predicts which predictor to use. Based on the result of the choice predictor, the global predictor or the local predictor predicts TAKEN or NOT TAKEN. During the training, the global history and the local BHT are updated by shifting in new history to the least significant bit position. The global PHT and the local PHT are updated based on the real outcome and the state transition. The choice PHT is updated by shifting to the predictor which is "more correct" based on the predictors' result and real outcome.

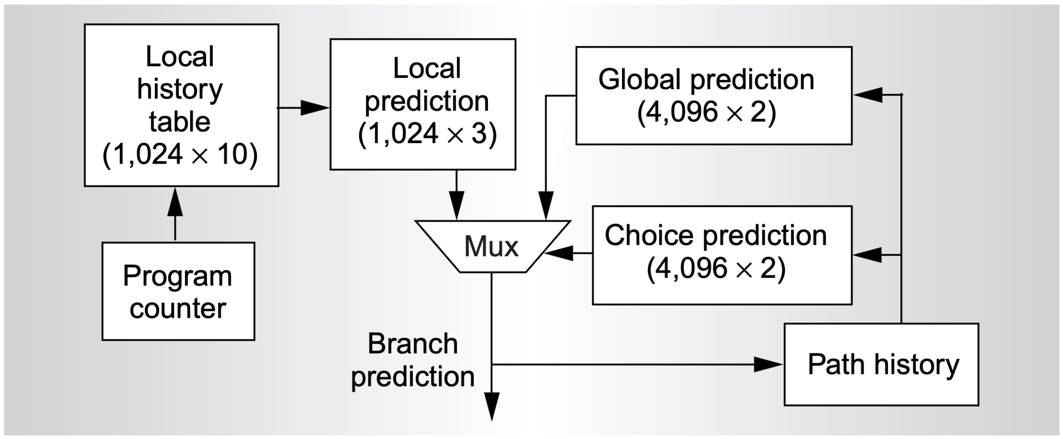
Local history table (1,024 × 10)
Local prediction (1,024 × 3)
Global prediction (4,096 × 2)
Program counter
Mux
Choice prediction (4,096 × 2)
Branch prediction
Path history

Figure 2: Tournament branch predictor architecture. image credit: The alpha 21264 microprocessor [2]

Compared with the G-share branch predictor, which only uses global history, the tournament branch predictor used in the alpha 21264 microprocessor [2] also takes the local branch history into consideration. This will make the predictor capture more local patterns. A choice predictor will also let the Tournament branch predictor capture more features and makes better use of global and local predictors.

However, the memory space used for the predictor is much higher than the G-share branch predictor. Also, the global predictor only uses global history, which means it may lack some information from the program counter.

2.3 Custom Branch Predictor

In this experiment, I combine the tournament branch predictor and the G-share branch predictor to form the custom branch predictor. As I stated in the previous subsection, the global predictor and the choice predictor of the tournament branch predictor only use global history. It means they only depend on the branch history patterns and may lack some information from the branch address. To fix it, I change the address indices of both predictors from only global history to XORs of global history and program counter, which is inspired by the G-share branch predictor.

This custom branch predictor is a combination of the tournament branch predictor and the G-share branch predictor. Hence, it is expected to inherit the advantages of both predictors. In other words, the custom predictor is expected to capture any correlation between different branches in making the prediction and local patterns based on branch addresses. It is expected to solve part of the problems the tournament branch predictor has, but the memory space used is still high.

To count the total amount of memory used for the custom predictor, I need to count the amount of memory used for the global predictor, the local predictor, and the choice predictor. For the global predictor, the size of the global PHT is $2 \times 2^{\text{ghistoryBits}}$ bits and the size of the global history register is ghistoryBits bits. For the local predictor, the size of the local BHT is $\text{lhistoryBits} \times 2^{\text{pcIndexBits}}$ bits and the size of the local PHT is $2 \times 2^{\text{lhistoryBits}}$ bits. For the choice predictor, the size of the choice PHT is $2 \times 2^{\text{ghistoryBits}}$ bits. In my experiment, ghistoryBits = 13, lhistoryBits = 11, and pcIndexBits = 11. So the total memory used is

$$2 \times 2^{13} + 13 + 11 \times 2^{11} + 2 \times 2^{11} + 2 \times 2^{13} = 59405 \text{ bits } < (64K + 256) \text{ bits}$$

3 Results

The table 1 shows the misprediction rate (in percent) of G-share, tournament, and custom branch prediction in six traces. Notice that the performance of the custom branch predictor is better than any of the other two predictors in all six traces, the custom branch predictor achieves the advantages that I expected in the design phase. As illustrated in the previous section, the custom branch predictor inherits the advantages of both predictors. In other words, the custom predictor captures correlations between different branches and local patterns based on branch addresses. The g-share branch predictor is successful in getting correlations between different branches, so it performs better than the tournament branch predictor in fp_1 and fp_2 traces. On the other hand, the tournament branch predictor also focuses the local patterns, which results in a better performance in the rest 4 traces compared with the g-share branch predictor.

trace	custom:13:11:11	gshare:13	tournament:9:10:10
fp_1	0.824	0.825	0.991
fp_2	0.387	1.678	3.246
int_1	9.675	13.839	12.622
int_2	0.273	0.420	0.426
mm_1	1.120	6.696	2.581
mm_2	6.823	10.138	8.483

Table 1: The misprediction rate (in percent) of G-share, tournament, and custom branch prediction in six traces

4 Conclusion

In this project, I explore the use of three different branch predictors, the G-share branch predictor, the tournament branch predictor, and a custom branch predictor. Both the G-share branch predictor and the tournament branch predictor have significant advantages and disadvantages. My custom branch predictor, a combination of two predictors, tries to inherit advantages and avoid disadvantages. Based on the experiment result, the custom branch predictor successfully outperforms Both the G-share branch predictor and the tournament branch predictor.

5 Contribution

Boyuan Tong finished the project solely. Boyuan Tong wrote the codes for all three branch predictors and wrote the entire report.

References

[1] P. Pannuto, "Branch predictor," *Lecture Slides*, 2023.

[2] R. E. Kessler, "The alpha 21264 microprocessor," *IEEE micro*, pp. 24–36, 1999.