Final Project Report: Branch Predictor

Chavisa Thamjarat PID: A53307009

1. INTRODUCTION

In modern processors, the Pipeline mechanism is widely adopted to improve the hardware performance. Even though the pipeline mechanism can accelerate the execution time by splitting one instruction into small stages on independent hardware components, it still could not fully optimize the hardware capability when there is a control dependence between instructions. The processor needs to stall until the branch condition is resolved and the next instruction is known before executing the next instruction. Later, the idea of Branch Speculation was developed. Branch Speculation basically assumes the next instruction address and starts to fetch it right after the previous instruction without any stall. However, this leads to a Control Hazard which occurs when the pipeline makes wrong decisions on branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded. Moreover, it also needs a Branch Recovery process which consumes an additional cycle and results in overall performance degradation.

To resolve the Control Hazard which is one of the major bottlenecks on a processor, a Branch Predictor is adopted to accurately predict the next instruction. Some basic approaches rely on the pattern of outcomes (taken/not taken) or the PC address to look up the history table and return the most recent outcome as a prediction. Another common approach is a two-bit predictor which gradually trains the predictor instead of fully relying on the latest outcome by adding a weak-taken and weak-not-taken state. Incorporating the idea of correlated and nearby branches, there are some works around Hybrid Model which combines two or more different approaches together.

Several factors influence the accuracy of the Branch Predictor including the frequency of instructions being taken, and the correlation between nearby branches. These factors determine what trade-offs are important for a given branch prediction engine, and by understanding these factors we can design branch predictors for specific classes of problems. In this report, two standard mechanisms, G-Share and Tournament, and the customer mechanism are experimented on six traces and the result is reported in section 3. This project is implemented by individual, not in team.

2. IMPLEMENTATION

2.1 G-Share

G-Share has evolved from G-Select which simply searches the branch history table or BHT by using the value in the branch history register as an index. The branch history register keeps the m-bit outcome pattern of all branches and assumes the correlations between the current branch and the other branches to make a prediction. It searches for repetitive patterns of global branch outcomes rather than keeping record of the specific branch histories. The BHT stores two-bit predictions represent four possible states being Weak Not-taken, Strong Not-taken, Weak Taken and Strong Taken. This means we need two consecutive Taken or Not-taken outcomes to change the prediction for each index in BHT.

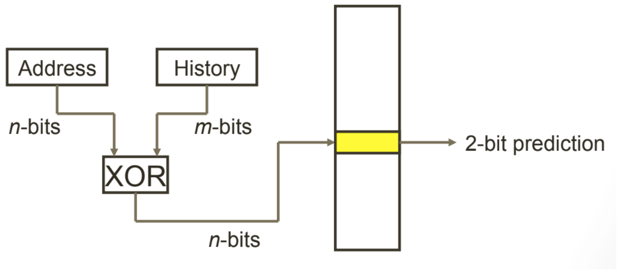

Figure 1: G-Share Predictor

The G-Share predictor also relies on BHT but it is indexed by the global branch history register XOR with a PC address. Consequently, one pattern can be mapped to different indices in the BHT. This technique reduces the collision between two similar patterns produced by different blocks of code.

Though G-Share takes PC as an input when hashing an index, it is still not as accurate as Local Branch predictor that directly uses PC address as an index but the gap is small. However, compared to the Local Branch predictor, G-Share is potentially faster since it requires only one memory access while the Local Branch predictor needs two accesses. (one to PC history and one to BHT)

2.2 Tournament

Tournament method takes advantage of both the global branch and local branch. The main idea is to combine both approaches and train the chooser to know when to use the global or the local prediction. It is composed of three components. First, the Global predictor which determines the prediction purely based on the global history of outcome. It

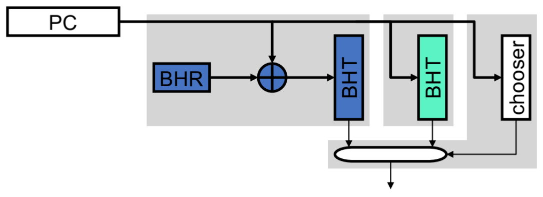

Figure 2: Tournament Predictor

searches the two-bit prediction from BHT using the global outcome pattern as an index. The second component is the local predictor. It first queries the local outcome pattern in the PC history table indexed by lower n-bit of PC address. Then the local outcome is further fed into the BHT to look up the two-bit prediction. Lastly, the chooser indicates which predictor to use for each time by basically counting how many times the global predictor beats the local predictor. In this mechanism, the chooser will increase its value when the global predictor beats and decrease when it loses. Tournament will always return a final prediction from the chosen predictor.

Different from a single predictor approach, this mechanism splits the BHT into two separate tables which keep tracking two different branch histories. Accordingly, it enhances the capability of the predictor to be aware of both global and local. Also, it can be adaptive to more favorable situations based on the chooser.

2.3 Custom Predictor

The customer predictor is composed of the G-Share predictors, Global predictor and Local predictor (similar details to the Tournament implementation). All three predictors will first make a prediction, then the final prediction is then subject to majority voting. If all three predictors do not agree, the prediction with two votes is what is trusted. If the G-Share predictor disagrees with the global predictor, we presume that there is a collision in the table,and therefore the local predictor, which takes PC as an input to search for a local pattern, will be the deciding vote.

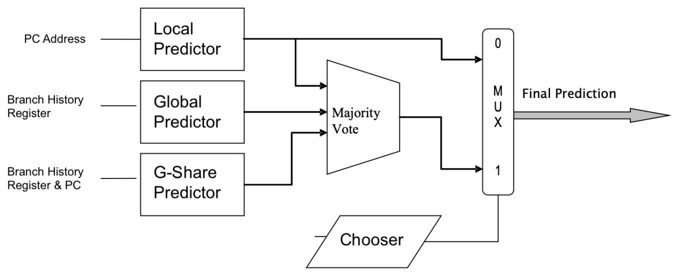

Figure 3: Custom Predictor

We also modify the idea from the Tournament approach that uses a concept of Chooser to choose the more likely predictor for each round. The Chooser always starts from selecting the local predictor then keeps tracking the number of times that the voting result beats the local predictor. Once the Chooser reaches the threshold, here we use the number of history bits, we will switch to use voting results. Also, when the local predictor beats voting results, we will switch to use the local predictor. The components of the custom predictor are illustrated in Figure 3.

3. OBSERVATION

In the experiment, we ran three predictors on six different traces then compared the misprediction rate. The misprediction rate is calculated from ratio between the number of instruction that was mispredicted and all instructions.

The configuration of the custom predictor is listed below.

G-Share Predictor

- Global History Bits 13

Tournament Predictor

- Global History Bits 9
- Local History Bits 10
- PC Index Bits 10

Custom Predictor

- G-Share Global History Bits 9
- Tournament Global History Bits 9
- Tournament Local History Bits 10
- PC Index Bits 10

	Misprediction Rate		
Traces	**GSHARE**	**TOURNAMENT**	**CUSTOM**
int_1	**11.680**	17.221	14.307
int_2	0.480	0.648	**0.464**
fp_1	1.156	1.241	**1.073**
fp_2	8.781	6.427	**3.553**
mm_1	5.939	10.822	**3.792**
mm_2	**8.219**	10.202	8.711

Figure 4: Misprediction Rate of G-Share, Tournament and Custom predictors on six traces.

From figure 4, the custom predictor has a lower misprediction rate than the Tournament predictor for all traces. The custom predictor also beats G-Share for four traces but loses on int_1 and mm_2. However, the gap of misprediction rates between both mechanisms is quite small. (11.7% vs 14.3% and 8.2% vs 8.7%) Additionally, there is an exceptional improvement in misprediction rate for fp_2 and mm_1 from the customer predictor compared to the other two. For fp_2, customer predictor decreases the misprediction rate by 36% from G-Share and 65% from Tournament. For mm_1, customer predictor decreases the misprediction rate by 60% from

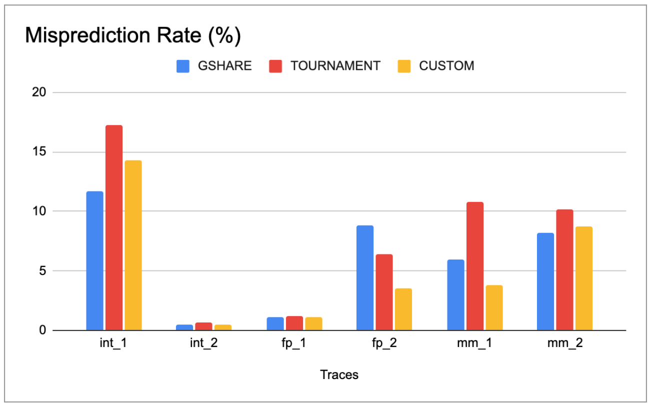

Figure 5: Misprediction Rate of G-Share, Tournamentand Custom predictors on six traces.

G-Share and 45% from Tournament.

From the experiment, we observe that there is no absolute winner or loser for all test traces. Even though Tournament has never won, it still beats G-Share once for fp_2. Hence it can be inferred that each of these Branch prediction techniques is capable to solve a different problem in branch prediction. While all predictors in this study use both global and local history knowledge to some extent, Tournament and custom predictors have more information than G-Share since it has the specific table contributing to the local branch and has a logic to select the appropriate predictors based on the recent accuracy. As expected, the custom predictor performs better than G-Share on average. However, the Tournament performs worse than the G-Share. This might be because the Tournament has a smaller bit size of history than the G-Share. If we consider the hardware constraint, G-Share predictor takes fewer hardware to implement and fewer access to the registers, but still results in a competitive misprediction rate compared to the custom one.

4. RESULTS AND CONCLUSION

In this project, we have implemented three branch predictors and evaluated them on six traces. The first predictor is G-Share which takes PC XORed with global history as an input to BHT which is light-weighted in terms of hardware and data access. The second mechanism is Tournament predictor which has more sophisticated logic to choose between local and global prediction. Finally, the custom one is the combination of G-Share and Tournament with the voting mechanism to choose which prediction is more accurate. The evaluation metric is the misprediction rate which is simply a ratio between misprediction and all instructions in the program. The result showed that the custom predictor outperformed on four traces and G-Share outperformed two traces. Additionally, the custom predictor performed exceptionally well on fp_2 and mm_1 compared against the other predictors. Though the result has shown that the custom design can eventually improve the performance of the processor in terms of branch prediction accuracy in most cases, this experiment does not give a well-rounded conclusion that considers other factors such as memory cost and computation time. To conclude the tradeoffs between each mechanism, other aspects of a Branch Predictor performance needs to be further studied.

5. REFERENCES

Alano, Lisa & Boven, Brad & Terrel, Andy. (2021). Dynamic Branch Prediction Study Combining Perceptrons and Bit-Counter Predictors.

Lecture slides of CSE 240A:Principles in Computer Architecture, Branch Prediction and Hardware Multithreading.