Branch Predictor

Chen Fu Huichen Ma

1 Introduction

In computer architecture, the branch predictor is a digital circuit that tries to guess which way the branch (for example, if-then-else structure) will go before knowing the branch explicitly. The purpose of the branch predictor is to improve the flow in the instruction pipeline.

Branch predictors play a key role in achieving high-efficiency performance in many modern pipelined microprocessor architectures such as x86. The branch predictor tries to avoid this waste of time by trying to guess whether it is most likely to use or not to use a conditional jump. In the best case, the branch predictor makes correct predictions, and we save a lot of time without waiting for the entire branch to complete.

There are two types of branch prediction schemes. The first type is static branch prediction: prediction is made at compile time, based on instruction type or analysis. It is suitable for branches that are easy to predict, such as for loops. The other is dynamic branch prediction: making predictions at runtime and using hardware to track runtime behavior. It is more accurate than static branch prediction, but dynamic branch prediction requires more chip area and power consumption.

Like any other element, branch prediction does not rely on a single component. The final prediction at the end of the loop is the final result of the direction predictor and the branch target buffer (BTB). Usually, the direction predictor uses a small state machine table, usually 2 bits (4 states), called the pattern history table (PHT). The difference between different predictors is usually the indexing scheme of PHT. BTB is a table, similar to a cache in design, storing the mapping of branch PCs and the target PCs they use.

- **Advantages and Disadvantages** The advantage of the branch predictor is that we can reduce the waste of pipeline time caused by incorrectly running branches by predicting branch operations in advance, thereby improving the efficiency of program operation. Of course, this also comes at a price. An accurate branch predictor usually requires additional operating resources and storage space. In addition, there will be branch prediction errors. In this case, speculative instructions are wasted. Performance may be the same as serialized execution.
- **Current Trend** Since the branch predictor is an energy-wasting speculation technique, its use in highly energy-constrained systems (such as battery-powered mobile devices) is challenging. In order to reduce the energy consumption of the branch predictor, it can be designed as a non-volatile memory, or it can use active energy management techniques such as power gating. In addition, the development of techniques for improving the accuracy of branch predictors and reducing the energy waste of wrong path execution will be a major research challenge for computer architects[1].

In this project, we mainly implemented three kinds of branch predictors -G-share, Tournament, and customized one. For the first two methods, we explored the impact of ghistoryBits, lhistoryBits, pcIndexBits on the performance of the predictor. For the last one, we try to use deep learning to build a method that can beat the first two predictors on all test examples. Finally, we conducted a simple analysis and comparison of the three methods and reached a conclusion.

2 Implementation

Before introducing the implementation of all branch predictors, we first introduce two-level pattern history table and n-bit saturating counters. They are common components of the classic branch predictor, and are also used in g-share and tournament methods.

Figure 1 is the two-level pattern history table, which uses the results of the most recent n-bit branch history as the index, stores the corresponding prediction results, and outputs them. When updating, it will update the prediction result of the corresponding index according to the corresponding actual branch result.

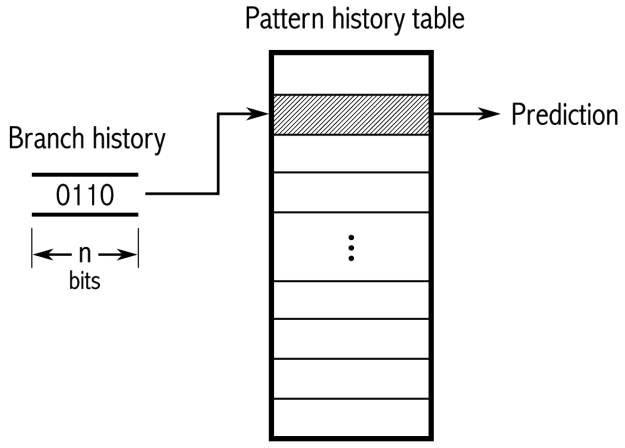

Figure 1: Two-level Pattern History Table

Due to the requirements of the project, we use 2-bit saturating counters when updating the two-level pattern history table[2]. It is divided into four states: strong taken, weak taken, weak not taken and strong not taken. If the pattern

is weakly not taken or strongly not taken, then we decide to predict the branch as "NOT TAKEN". Otherwise, the prediction will be "taken". According to the branch result, the state transition is performed and the corresponding history table is updated. The state transition process is shown in figure 2.

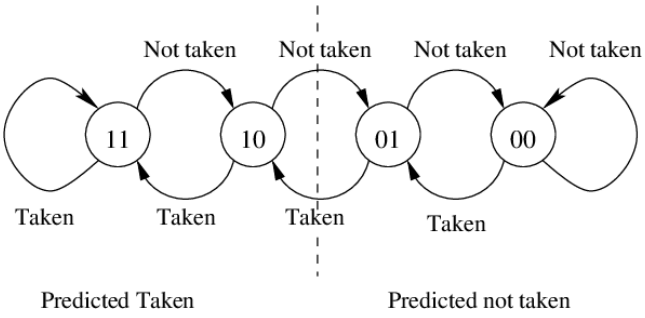

Figure 2: Two-Bit Saturating Counter

Next, we will give a specific introduction to different branch predictors.

2.1 G-Share

G-share is a global shared branch predictor. It uses the global branch history and the location of the branch instruction(PC) to create the index of the pattern history table. The whole process is shown in Figure 3.

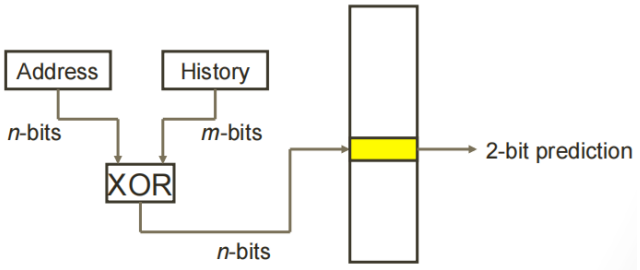

Figure 3: G-share Implementation

Here, we XOR the global branch history (the length of which is determined by the input parameters) and the lower bits of branch address (PC) as the index of the pattern history table, and use the 2-bit saturating counters component introduced before to make predictions and update. In addition, we also need to update the global branch history by moving the new branch result to the last bit and remove the first bit.

- **History** G-share predictor is a kind of dynamic branch predictor as mentioned before. The Bimodal Predictor is an early dynamic branch predictor. It was created to easily predict branches that exhibit consistent taken or not-taken behavior. This predictor uses only local (PC-specific) branch resolution and is susceptible to interference from other branches indexed to the same entry. The next evolution of branch prediction is the global history branch predictor, which uses a global pattern history register. The new scheme uses the low-level log2n bits of the global history register to index into the table to find the correlation between the previously executed branch and the current prediction. And then we get our G-share predictor, which makes use of both temporal and spacial attributes of the branch and achieves great success[3].

- **Advantages and Disadvantages** The advantage of using G-share is that it has more contextual information. To be specific, it uses the global history table which is impacted by the thought “recently executed branch outcomes in the execution path is correlated with the outcome of the next branch Global branch correlation”[4]. It also makes better use of the pattern history table. However, because of the introduction of the XOR operation, it increases the access latency compared to only using the global history.

2.2 Tournament

The tournament predictor is a hybrid predictor. The main idea of the hybrid branch predictor is to use two or more predictors and combine these results in some way to obtain higher accuracy. Upon request, we implemented a tournament predictor similar to Alpha 21264[5]. The Alpha 21264 tournament consists of two different predictors: a local predictor and a global predictor. Its basic structure is shown in Figure **??**.

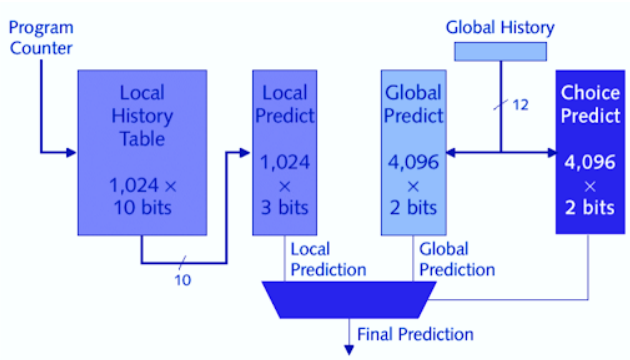

Figure 4: Tournament Implementation

The local predictor is a 2-level predictor that contains a branch history table and a pattern history table. Unlike the structure of the Alpha 21264 predictor, we use a 2-bit saturating counters. The predictor first determines the index of the branch history table based on the input PC information and pcIndexBits, and then looks up the history information (taken, not-taken) of the corresponding branch based on the index. And then based on lhistoryBits, the previously introduced pattern history table and 2-bit saturating counters module makes predictions and updates. In addition to updating the pattern history table, we also need to update the actual branch result to the branch history table.

The global predictor uses the global branch history to index the pattern history table. The structure of this part is basically the same as that of g-share, except that we don’t need to consider the PC information at this time, but only need to use the corresponding length of the global branch history as the index of the pattern history table according to ghistoryBits.

In addition, we also need a chooser to select the above two different branch predictors. Its implementation is similar to

the global predictor. We still regard the global branch history of the corresponding length as the index of the pattern history table. At this time, the information stored in the pattern history table is not the prediction information for taken and not-taken, but which predictor to choose. Specifically, when the 2-bit saturating counters value is 0, 1, we choose the global predictor; when the value is 2, 3, we choose the local predictor. The update of chooser needs to depend on the prediction correctness of two different predictors, and the update method is shown in Figure 5. The idea is that according to the index of different global history, the higher the accuracy of which predictor predicts, the more possible it is to choose that predictor[4].

P1-correct	P2-correct	P1-correct - P2-correct	Action
0	0	0-0 = 0	None
0	1	0-1 = -1	Decrement
1	0	1-0 = 1	Increment
1	1	1-1 = 0	None

Counter value	Use predictor
00	P2
01	P2
10	P1
11	P1

Selects which predictor table to use for the prediction

Figure 5: Updating method of Chooser

- **History** Tournament predictor is also a kind of dynamic branch predictor. The idea of combination is first proposed by McFarling[4]. Since all predictors like Gshare can be configured to consume roughly the same on-chip space and run in the same clock cycle, McFarling suggests that we can combine the advantages of several predictors. The tournament predictor described in McFarling's paper was originally designed to choose from two branch predictors: local and global and achieved great success.
- **Advantages and Disadvantages** The advantage of the tournament is that it can combine the advantages of different branch predictors to achieve better accuracy. It can also reduce the warm-up time. However, it also has several disadvantages. It requires a meta-predictor or selector and has a longer access delay.

2.3 Custom Predictor

To improve prediction performance, we introduced the perceptron to our custom predictor. For each branch, it will be assigned to one specific perceptron to predict its outcome. The perceptron is a simple binary classifier like Neural Network with a single neuron as Figure 6 shows.

So how does the perceptron predictor work? As shown in Figure 6, perceptrons get the history information to make future prediction, where x_i represents the bits of a global branch history shift register, and w_i are corresponding weight values. Figure 7 shows the process of how to train a perceptron. It is an intuitive method because we just increase the weight when there is a positive correlation and decrease when negative. Weights with large magnitude should make more influence on

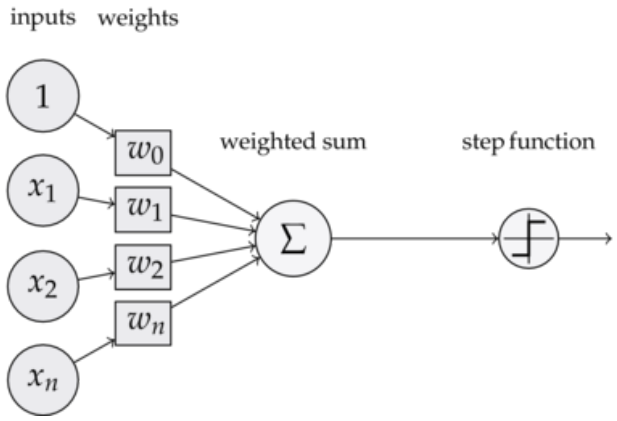

Figure 6: What exactly is a perceptron?

$$\begin{aligned}&\texttt{if } \text{sign}(y_{out}) \neq t \texttt{ or } |y_{out}| \leq \theta \texttt{ then}\\ &\quad \texttt{for } i := 0 \texttt{ to } n \texttt{ do}\\ &\quad\quad w_i := w_i + t x_i\\ &\quad \texttt{end for}\\ &\texttt{end if}\end{aligned}$$

Figure 7: Training Perceptrons

prediction. The threshold θ is a parameter to decide whether there is enough training. According to [6], the best threshold θ should be $\lfloor 1.93h + 14 \rfloor$ where h represents the number of history record bits.

Now we will be putting it all together to show how the whole pipeline is going in the prediction process. As shown in Figure 8, the following steps are conceptually taken [6]:

1 The branch address is hashed to produce an index $i \in 0..N-1$ into the table of perceptrons.

2 The i^{th} perceptron is selected from the table into a vector register $P_{0..n}$ of weights.

3 The value of y is computed as the dot product of P and the global history register.

4 The branch is predicted not taken when y is negative, or taken otherwise.

5 Once the actual outcome of branch becomes known, the training algorithm uses this outcome and the value of y to update the weights in P.

6 P is written back to the i^{th} entry in the table.

- **History** Before this method occurred, branch prediction focused on eliminating aliasing in two-level adaptive predictors, which used tables of saturating counters [7] [4]. With the development of machine learning, we were offered another possibility to increase the prediction accuracy. Perceptron is the simplest model of machine learning and does not take much memory to train.

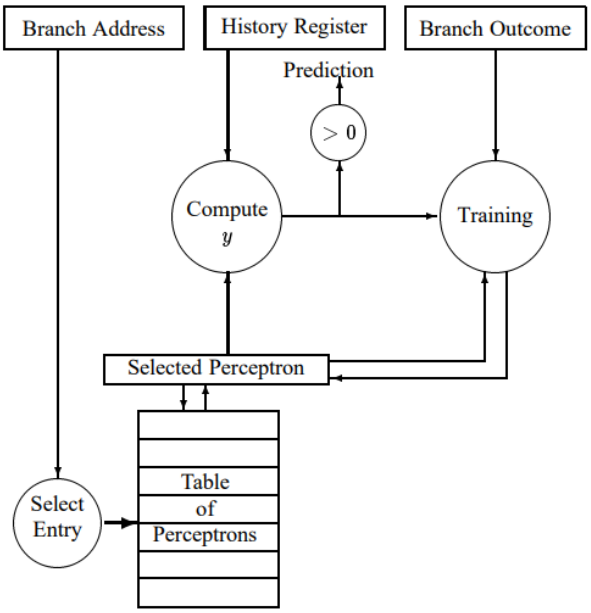

Figure 8: Perceptron Predictor Block Diagram

Table 1: Misprediction rate of Gshare

Para	fp_1	fp_2	int_1	int_2	mm_1	mm_2
9	1.466	10.733	26.114	0.968	16.646	15.157
11	1.174	5.393	19.049	0.583	9.891	11.912
13	0.825	1.678	13.839	0.420	6.696	10.138

So [6] explored the use of perceptrons and presented this new predictor.

- **Advantages and Disadvantages** As a machine learning model, perceptron is rather easy to be implemented and takes really little resources to train. Requiring linear resources means that perceptron predictor is able to consider longer history length and achieve better prediction accuracy with less training resources.Limited to its linear functionality, perceptron predictor could only learn linearly inseparable functions. Another potential drawback is the increased computational complexity.

3 Observation

3.1 G-Share

After implementing the g-share method, we tested 6 branch prediction test files and tried to explore the influence of different ghistoryBits on the results. Specifically, we tested all tests for the cases where ghistoryBits is 9, 11, and 13. The results obtained are shown in the figure 9 and table 1.

We analyze the above data and table information. In general, the misprediction rate decreases as the table size and history length increase. Therefore, the number of bits is an important factor in Gshare. The longer the global history register, the stronger the correlation between the global pattern and the current branch prediction can be found by the predictor. If we want to have a better prediction, we should

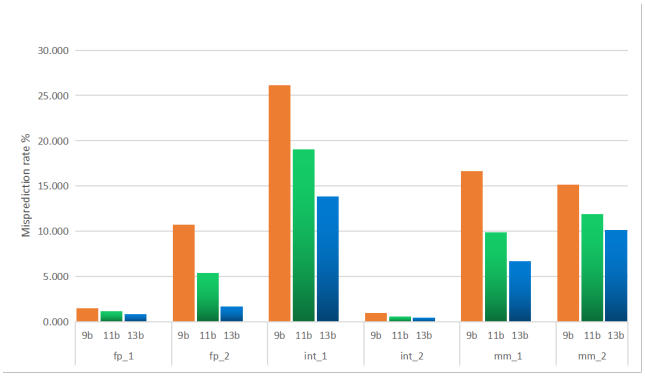

Figure 9: Gshare - ghistoryBits

Table 2: Misprediction rate of Tournament - ghistoryBits

Para	fp_1	fp_2	int_1	int_2	mm_1	mm_2
5,10,10	1.091	3.730	15.096	0.465	4.018	9.237
7,10,10	0.990	3.640	14.292	0.448	3.641	8.996
9,10,10	0.991	3.246	12.622	0.426	2.581	8.483

keep more bits of history. Specifically, more bits means more Gshare information. Each time ghistoryBits increases by 1, the history length increases by 1 bit. In addition, the number of bits used to index from the PC to the table has also increased by 1. More local PC address information and global history information provide more reference and help for the prediction of branch prediction, thereby improving the accuracy of prediction. This also explains why the 13-bit branch predicts the best.

3.2 Tournament

After implementing the tourism method, we tested on 6 test files and tried to explore the influence of ghistoryBits, lhistoryBits, pcIndexBits on the prediction accuracy. Specifically, we tested the tournament models with ghistoryBits of 5, 7, 9; lhistoryBits of 6, 8, 10; pcIndexBits of 6, 8, 10, and the results obtained are shown in the figure 10, 11, 12 and table 2, 3, 4.

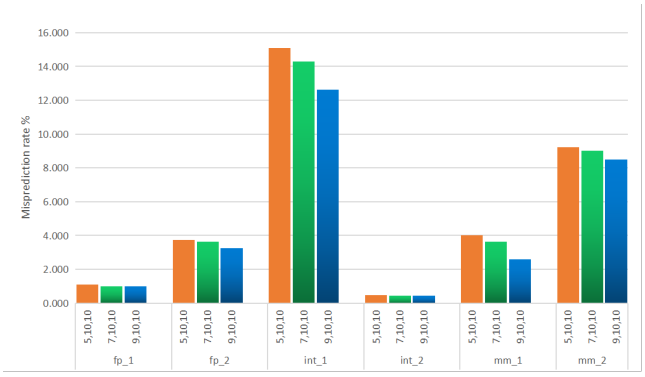

Figure 10: Tournament - ghistoryBits

Table 2 shows the experiment about ghistoryBits. According to the data analysis of the table, it can be seen that with

Table 3: Misprediction rate of Tournament - lhistoryBits

Para	fp_1	fp_2	int_1	int_2	mm_1	mm_2
9,6,10	0.992	2.822	13.324	0.505	7.286	9.018
9,8,10	0.991	3.083	12.877	0.460	5.093	8.698
9,10,10	0.991	3.246	12.622	0.426	2.581	8.483

Table 4: Misprediction rate of Tournament - pcIndexBits

Para	fp_1	fp_2	int_1	int_2	mm_1	mm_2
9,10,6	1.179	3.828	18.643	0.649	7.705	12.352
9,10,8	0.997	3.637	15.745	0.476	4.502	10.385
9,10,10	0.991	3.246	12.622	0.426	2.581	8.483

the increase of ghistoryBits, the prediction accuracy rate continues to rise. Due to the requirements of the topic, in the implementation of the tournament, ghistoryBits represents the input size of the global and choice predictors. Just like the analysis in the g-share predictor, more global history can provide more contextual information to help improve prediction accuracy. Similarly, historical information can also help chooser to choose a suitable predictor, so as to better combine two different predictors and improve the accuracy of prediction.

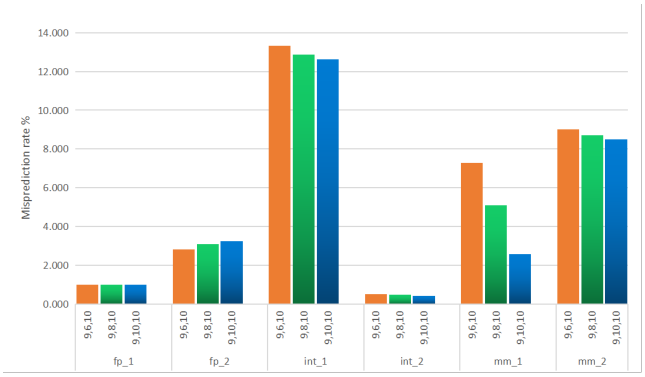

Figure 11: Tournament - lhistoryBits

Table 3 shows the experiment about lhistoryBits. According to data analysis, generally speaking, with the growth of lhistoryBits, the prediction accuracy of the predictor is improving. However, this is not absolute. For fp_1, this trend is not significant; for fp_2, the prediction accuracy rate is on the contrary declining. Correspondingly, this trend of change is most prominent on mm_1, and its prediction accuracy rate has risen significantly. lhistoryBits represents the historical information of each branch. The more historical information, the more accurate we can predict its future behavior. However, for some branches with fewer occurrences or branches with elusive behaviors, the use of more historical information will cause over-interpretation.

Table 4 shows the experiment about pcIndexBits. According to the data analysis of the table, it can be seen that with the increase of pcIndexBits, the prediction accuracy rate continues to rise. pcIndexBits characterizes that when local history is used for prediction, the index length of different branches is recorded. Obviously, the larger the pcIndexBits, the more

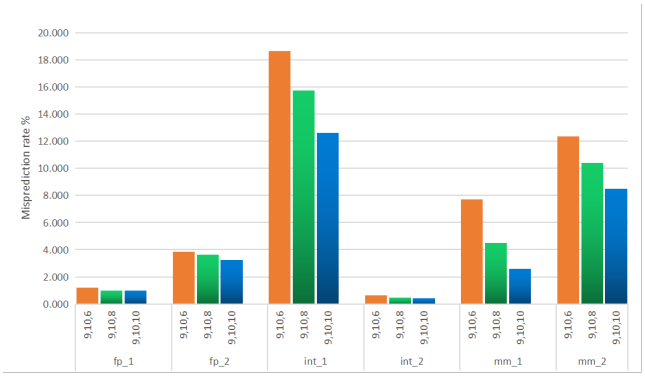

Figure 12: Tournament - pcIndexBits

Table 5: Misprediction rate of Perceptron Predictor

fp_1	fp_2	int_1	int_2	mm_1	mm_2
0.823	0.963	7.922	0.299	1.876	7.396

detailed branch information it records, and the more accurate the division of different branches. However, too long pcIndexBits will take up a lot of memory, so you need to set pcIndexBits according to the conditions.

3.3 Custom Predictor

For each address trace, we tested our Custom Predictor (Perceptron) and Table 5 shows the results of misprediction rate. This predictor does make a great improvement on branch prediction.

The only requirement is that the total size of the custom predictor must not exceed $(64K + 256)$ bits (not bytes) of stored data. Considering this limitation, we choose 248 as the number of perceptrons and 32 bits of weight. That also means 32 bits for global branch history record. Therefore our total number of bits needed is $(32 + 1) \times 8 \times 248 + 32 = 65504$ bits, which is less than the budget $64K + 256 = 64 \times 1024 + 256 = 65792$ bits. And the most important is that our custom predictor does outperform both the Gshare and Tournament predictors (details below).

3.4 Comparison among Branch Predictors

As shown in Table 13 and Figure 13, the custom predictor (Perceptron) outperforms both the Gshare and Tournament predictors in all 6 traces. We would like to make further explanation about why does it work well. The main advantage is due to its ability to use longer history length. Either Gshare or Tournament requires exponential space resources in the history length, while perceptron predictor just requires linear resources. That is why the perceptron predictor could achieve

Table 6: Comparison Between G-share 13, Tournament 9,10,10, and Custom Method

Type	fp_1	fp_2	int_1	int_2	mm_1	mm_2
Gshare	0.825	1.678	13.839	0.420	6.696	10.138
Tourna	0.991	3.246	12.622	0.426	2.581	8.483
Custom	0.823	0.963	7.922	0.299	1.876	7.396

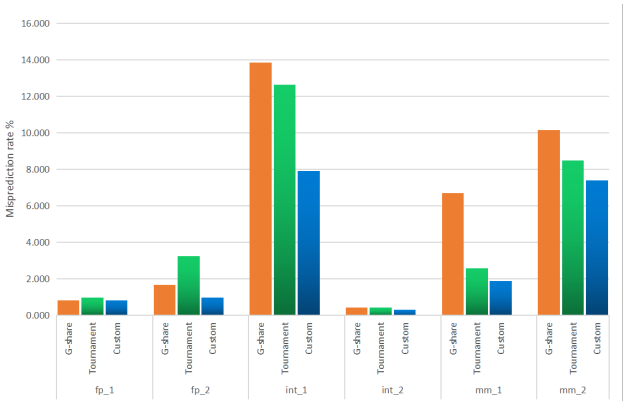

Figure 13: Misprediction rate of different predictor

higher accuracy on branch prediction.

4 Results and Conclusion

In this project, we implemented three methods - Gshare, Tournament, Perceptron - for branch prediction and tested each of them on all six traces. We analyzed the performance results and gave reasonable explanation about it.

Through our experiments and analysis, we concluded that the number of bits is an important factor in branch predictor. The longer the global history register, the stronger the correlation can be found by the predictor. This also explains why does the perceptron predictor works better than the other two predictors. Perceptron predictor just requires linear resources, which makes it able to use longer history length, so as to improve the accuracy of prediction.

5 Distribution of Work

This is a challenging project finished by Huichen and Chen. As for Gshare and Tournament Predictor, Huichen implemented the code, performed results analysis and completed the corresponding report. Chen implemented the work of Custom Predictor including the code and report. For the rest of report, Huichen completed the introduction of literature review and Chen completed the results comparison and conclusion. Huichen also plotted graphs to make the analysis visually appealing.

6 References

[1] S. Mittal, "A survey of techniques for dynamic branch prediction," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 1, p. e4666, 2019.

[2] T.-Y. Yeh and Y. N. Patt, "Two-level adaptive training branch prediction," in *Proceedings of the 24th annual international symposium on Microarchitecture*, pp. 51–61, 1991.

[3] J. E. Smith, "A study of branch prediction strategies," in *25 years of the international symposia on Computer architecture (selected papers)*, pp. 202–215, 1998.

[4] S. McFarling, "Combining branch predictors," tech. rep., Citeseer, 1993.

[5] R. E. Kessler, "The alpha 21264 microprocessor," *IEEE micro*, vol. 19, no. 2, pp. 24–36, 1999.

[6] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pp. 197–206, IEEE, 2001.

[7] C.-C. Lee, I.-C. Chen, and T. N. Mudge, "The bi-mode branch predictor," in *Proceedings of 30th Annual International Symposium on Microarchitecture*, pp. 4–13, IEEE, 1997.