Branch Predictor

Xingkai Zheng A59018348

ABSTRACT

A branch predictor can help the program decide to jump to the target or remain at the origin place when it meets branch instruction. In this work, I first built different components of branch predictors such as selector, global predictor, local predictor, etc. Then, I combined them together to make Gshare, Tournament, Perceptron, and Custom predictor. Thus, Gshare and Tournament predictor was validated. And the new custom predictor beat both Gshare and Tournament.

1. INTRODUCTION

Branch predictors are digital circuits used in computer architecture to improve the performance of instruction pipelines by guessing the outcome of branches, such as if-then-else structures before they are known definitively. These circuits are critical for achieving high performance in many modern pipelined microprocessor architectures, such as x86. Without branch predictors, the processor will lose its performance when waiting for the result of branch instructions. Although branch predictors can improve processor performance, An complex branch predictor may lead to extra overheads. Thus, when building branch predictors we should take storage and speed into consideration. Nowadays, people not only try to improve the structure of branch predictors but also upgrade their hardware implementation [1].

1.1 Gshare predictor

The GShare branch predictor [2] [3]uses a hashing method to make predictions. The instruction address and the GHR are combined using a simple XOR operation, and the resulting hash is used to index into a table of two-bit counters. This allows the GShare predictor to make efficient and accurate predictions, improving the performance of the instruction pipeline.

1.2 Tournament predictor

In a tournament predictor [4], two different prediction algorithms are maintained, one based on global information and one based on local information. The selection strategy determines which algorithm to use in a given situation. For example, the local predictor may be used, and when it is incorrect, the prediction can be switched to the global predictor. Alternatively, switching can only be done when there are two different predictions. Since 2006, tournament predictors using 30K or more bits have been used in processors such as Power5 and Pentium 4. These predictors can achieve better accuracy at moderate sizes (8K - 32K bits) and effectively utilize a large number of prediction bits.

1.3 Perceptron predictor

The perceptron branch predictor [5] uses a simple neural network, known as a perceptron, as an alternative to the commonly used two-bit counters. The perceptron is trained to learn correlations between the outcomes of previous branches in the global history and the behavior of the current branch. These correlations are represented by weights, with larger weights indicating stronger correlations and greater contribution to the prediction of the current branch. This allows the perceptron branch predictor to make more accurate predictions than traditional two-bit counters.

1.4 Selector

A selector can choose the output from two different branch predictors based on global history or Program Counter. When training the selector, it will turn to the right predictor when two predictors have different choices.

2. METHODS

2.1 Predictor Data Structures

Firstly, we should build the structure in the branch predictors. In detail, a Selector is composed of a global history, an index mask, a Choice history table, and left&right counter to count which side it chooses. Gshare and Global predictors include global history, a branch history table, and an index mask. Local predictor has a local history table, a local prediction table, a PC mask, and a local index mask. A tournament predictor consists of a local predictor, a global predictor, and a selector of the above two predictors. A perceptron predictor contains global history, an index mask, a perceptron table, a weight number, a bias theta, and a weight bound. Mask is used to simulate different storage of branch predictors. The above structures are shown in Table 1.

Table 1: Predictor Data Structures Table

Structure	Values
Selector	history, CHT, lcounter, rcounter, mask
Gshare	history, BHT, mask
Global	history, BHT, mask
Local	pcmask, lmask, lBHT, lPT
Tournament	G(Global),L(local),S(selector)
Perceptrons	history, mask, PLT, weightnum, theta, bound

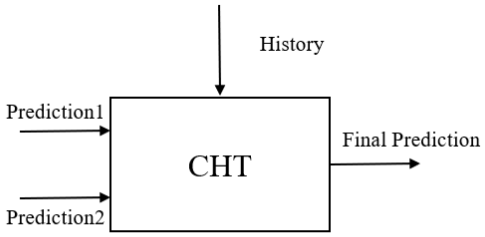

Figure 1: Selector

2.2 Implementation of Selector

The Structure of Selector is shown in Figure 1.

- **Init:** Firstly, the function initializes the global history and left counter & right counter to zero. Then, it assigns $Storage = 2^{ghistoryBits} * sizeof(uint8_t)$ to CHT and initializes CHT with a full of zero. Finally, it builds a mask full of "1" with the length of $ghistoryBits - 1$.
- **Pred:** Firstly, the function uses mask and history to get the index. Then, it uses the index to look up CHT and gets the prediction. If the prediction is bigger than 2, it means TAKEN, else NOT TAKEN. And the function adds 1 to the corresponding counter.
- **Train:** Firstly, the function uses the same way above to get the prediction. If the left prediction is wrong and the right prediction is right, then it turns the prediction a step to right. If the right prediction is wrong and the left prediction is right, then it turns the prediction a step to left. And if the prediction is out of bound after the move, the function replaces it with the bound number. Besides, it needs to add the outcome to history.

2.3 Implementation of Gshare Predictor

The Structure of Gshare Predictor is shown in Figure 2.

- **Init:** Firstly, the function initializes the global history. Then, it assigns $Storage = 2^{ghistoryBits} * sizeof(uint8_t)$ to BHT and initializes BHT with a full of zero. Finally, it builds a mask full of "1" with the length of $ghistoryBits - 1$.
- **Pred:** Firstly, the function uses mask and the result of $PC \oplus history$ to get the index. Then, it uses the index to look up BHT and gets the prediction. If the prediction is bigger than 2, it means TAKEN, else NOT TAKEN.
- **Train:** Firstly, the function uses the same way above to get the prediction. If the prediction is wrong and the outcome is TAKEN, then it adds 1 to the prediction. Else, if the prediction is wrong and the outcome is NOT TAKEN, then it subtracts the prediction by 1. And if the prediction is out of bound after the move, the function replaces it with the bound number. Besides, it needs to add the outcome to history by shifting history.

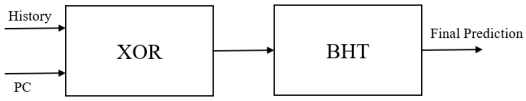

Figure 2: Gshare Predictor

Figure 3: Global Predictor

2.4 Implementation of Global Predictor

The Structure of Global Predictor is shown in Figure 3.

Except for the index calculation, the Implementation of Global Predictor is totally the same as Gshare Predictor. In Global Predictor, this function uses *history* instead of $PC \oplus history$ to get the index.

2.5 Implementation of Local Predictor

The Structure of Local Predictor is shown in Figure 4.

- **Init:** It assigns $Storage = 2^{pcIndexBits} * sizeof(uint16_t)$ to lBHT and initialize lBHT with a full of zero. besides, it also assigns $Storage = 2^{lhistoryBits} * sizeof(uint16_t)$ to lPT and initialize lPT with a full of zero. Finally, it builds a PC mask full of "1" with the length of $pcIndexBits - 1$ and a local history mask full of "1" with the length of $lhistoryBits - 1$.
- **Pred:** Firstly, the function uses the PC mask and the PC to get the PC index. Then, it uses the PC index to look up lBHT and get the local history. Next, it combines local history and local history mask to get the local index. Thus, the prediction is at $index^{th}$ place in lPT. If the prediction is bigger than 2, it means TAKEN, else NOT TAKEN.
- **Train:** Firstly, the function uses the same way above to get the prediction. If the prediction is wrong and the outcome is TAKEN, then it adds 1 to the prediction. Else, if the prediction is wrong and the outcome is NOT TAKEN, then it subtracts the prediction by 1. And if the prediction is out of bound after the move, the function replaces it with the bound number. Besides, it needs to add the new outcome to the local history ($lBHT[pcindex]$).

2.6 Implementation of Tournament Predictor

The Structure of Tournament Predictor is shown in Figure 5.

- **Init:** The function respectively initializes a Global predictor, a Local predictor, and a selector. The global

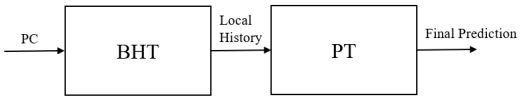

Figure 4: Local Predictor

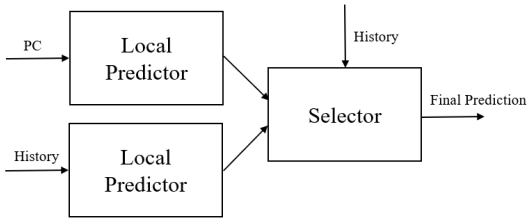

Figure 5: Tournament Predictor

bits of the Global predictor must be the same as the selector's according to the 21264 tournament branch predictor engineering realization [4].

- **Pred:** The function gets the local prediction and global prediction separately. Then, it takes these two elements as the input of the selector and gets the final prediction.

- **Train:** Firstly, The function uses the same way above to get the predictions. Then, it uses the outcome to train the global and local predictors by the above functions. Finally, it puts local prediction and global prediction and outcome into the selector train function to improve selector parameters.

2.7 Implementation of Perceptron Predictor

The Structure of the Perceptron Predictor is shown in Figure 6.

- **Init:** The function first initializes the history and mask of the perceptron structure with the specified values. It then allocates $memory = 2^{pcindexBits} * weightnum * sizeof(int8_t)$ for the PLT array using the "malloc" function and initializes all elements of the array to 0 using the "memset" function. The remaining fields of the perceptron structure such as *weightnum*, *bound* are also initialized with the provided values.

- **Pred:** The function first computes the index of the PC value in the perceptron's lookup table (PLT). It then calculates the result of the perceptron by iterating through the weights in the PLT and summing the weights that correspond to the current history of the perceptron. The function then returns the predicted outcome of the branch based on the sign of the result. If the result is positive, the function returns the "TAKEN" value, indicating that the branch is predicted to be taken. Otherwise, the function returns the "NOTTAKEN" value, indicating that the branch is predicted to be not taken.

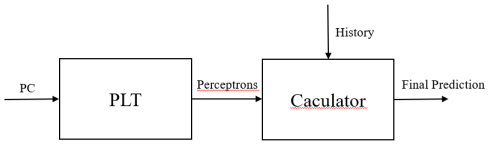

Figure 6: Perceptron Predictor

- **Train:** The function first computes the index of the PC value in the perceptron's lookup table (PLT). It then calculates the result of the perceptron by iterating through the weights in the PLT and summing the weights that correspond to the current history of the perceptron. If the outcome of the branch associated with the given PC value is different from the predicted outcome, the function adjusts the weights in the PLT accordingly. Finally, the function updates the history of the perceptron by shifting the current history one bit to the left and appending the outcome of the branch. Here the function uses bound to simulate the real bits of weights and uses theta to determine the little right result because the weight matrix is not significant enough and still needs to be trained.

2.8 Implementation of Custom Predictor

I designed 4 custom predictor which is the following and The Structure of the Custom Predictors are shown in Figure 7:

1. **Gshare+Tournament:** This means the method generates a Gshare predictor and a Tournament predictor. Then, it generates a selector to choose between the outcomes of the Gshare predictor and the Tournament predictor. In the init function, it just initializes Gshare, Tournament, and selector separately. In the pred function, it uses the selector to choose the answer between Gshare prediction and Tournament prediction. In the train function, it uses the same way above to get the predictions. Then, it trains the Gshare predictor and Tournament predictor with their corresponding predictions and outcome. Besides, it trains the selector with the three predictions and output.

2. **Gshare+Perceptron:** This method is the same as above Gshare+Tournament when you replace Perceptron with Tournament.

3. **Tournament+Perceptron:** This method is the same as above Gshare+Tournament when you replace Perceptron with Gshare.

4. **Gshare+Tournamnet+Perceptron:** This method can be seen as two iterations of the above method. It is shown below:

$$Methods = ((Gshare + Tournament) + Perceptron)$$

It first uses a selector to combine the results of Gshare and Tournament. And then it uses another selector to combine the above result and the Perceptron result.

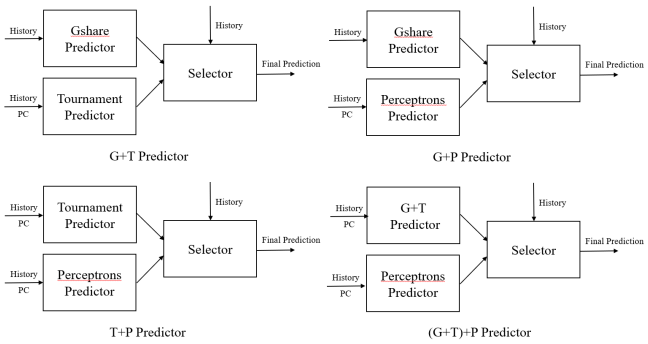

Figure 7: Custom Predictors

3. EXPERIMENTS

3.1 Experiment Parameters

The experiment parameters are shown below in table 2.

Table 2: Experiment Parameters Table

Parameter	Values
Selector global history bits	9
Gshare global history bits	13
Tournament global history bits	9
Tournament PC bits	10
Tournament Local history bits	10
Perceptrons weight num	12
Perceptrons PC bits	8
Perceptrons weight bound	16
Perceptrons weight theta	8

3.2 Memory Check

The following calculations ignore individual variables. When we calculate custom predictors, we choose the biggest one which is $Gshare + Tournamnet + Perceptron$.

- **selector:**

$$Memory \approx 2^9 * 2 = 1Kb << 64kb$$

- **Gshare:**

$$Memory \approx 2^{13} * 2 = 16Kb << 64kb$$

- **Tournament:**

$$Memory = Local + Global + Selector$$
$$\approx 2^{10} * 10 + 2^{10} * 2 + 2^9 * 2 + 2^9 * 2 = 14Kb << 64kb$$

- **Perceptron:**

$$Memory \approx 2^{pcindexbits} * weightnum * (log_2^{bound} + 1)$$
$$= 2^8 * 12 * 5 = 15kb << 64kb$$

- **Custom:**

$$Memory =$$
$$Gshare + Tournament + Perceptrons + Selector * 2$$
$$\approx 16k + 14k + 15k + 1k * 2 = 47Kb << 64kb$$

3.3 Experiment Results

The experiment results are shown below in table 3,4,5.

Table 3 shows the misprediction rates of independent predictors. In table 3, we can figure out that Gshare, Tournament and Perceptron all are not stable. Their prediction accuracies depend on the type of file a lot. Besides, because each of three predictors wins in different files. all three predictors have their advantages under nearly the same memory limit.

Thus, if we want to build a predictor to beat gshare & tournament, we should not just focus on independent predictors. This also proves that gshare & tournament are excellent predictors at moderate sizes.

Inspired by the design of Tournament Predictor, we can use selectors to let different predictors work on their areas of expertise. So We can combine the three above predictors together by selectors.

Table 3: Branch predictor Misprediction Rate Table

Files	Gsahre	Tournament	Perceptron
fp_1	**0.842**	0.994	0.955
fp_2	**1.500**	2.974	4.334
int_1	13.900	**12.630**	13.747
int_2	**0.426**	0.432	0.482
mm_1	6.523	**2.625**	4.711
mm_2	10.229	8.493	**8.389**

Table 4 shows the misprediction rates of custom predictors. In table 4, we test four different combinations including G+T, G+P, T+P, (G+T)+P. The results show a combination of two predictors performs better than anyone of them. And the combination of all three predictors performs best among all the custom predictors and independent predictors.

Table 4: Custom predictor Misprediction Rate Table

Files	G+T	G+P	T+P	(G+T)+P
fp_1	0.827	0.824	0.908	**0.820**
fp_2	1.315	1.497	1.543	**1.309**
int_1	10.931	11.623	11.109	**10.515**
int_2	0.352	0.381	0.354	**0.336**
mm_1	1.765	4.251	1.620	**1.570**
mm_2	7.993	7.985	7.463	**7.397**

Table 5 shows the right choice rate in the selector. For example, it is the rate when the selector chooses Tournament in G+T or Perceptrons in G+P T+P (G+T)+P. We need to look at Tables 3,4 and 5 in combination. For fp1 in G + T, we can see the selector only chooses the Tournament predictor at the rate of 9.6%. This is consistent with the result that the Gshare predictor performs best in Table 3. Similarly, for mm2 in G+P & T+P & (G+T)+P, the rates when the selector chooses the perceptron predictor are all above half, which means the perceptron predictor performs even better than G+T. Unsurprisingly, the perceptrons predictor works best for the mm2 file in table 3. All the facts above show the idea that combination enables different predictors to play the part of prediction they are good at is right.

Table 5: Custom predictor Right Choice Rate Table

Files	G+T	G+P	T+P	(G+T)+P
fp_1	9.6	8.3	83.9	27.9
fp_2	16.8	46.7	45.1	75.1
int_1	56.7	56.0	46.4	43.4
int_2	94.8	7.1	4.1	4.7
mm_1	40.0	62.5	54.9	67.8
mm_2	64.9	70.6	56.0	57.5

4. CONCLUSION

I first built components of different predictors (some predictors may be good components for other more complex predictors). Then, I built Gshare, Tournament, and Perceptron predictors using components. After I did some experiments both in accuracy and selector choice, I gets the following results. When faced with different files, the prediction accuracies displayed by the perception, tournament, and gshare predictors are different. And a combination of different types of predictors could improve the accuracy. Thus, the Custom (G+T)+P performs best in all six files.

5. REFERENCES

[1] P. Trivedi and S. Shah, "Reduced-hardware hybrid branch predictor design, simulation analysis," in *2019 7th International Conference on Smart Computing Communications (ICSCC)*, pp. 1–6, 2019.

[2] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. Patt, "Branch classification: A new mechanism for improving branch predictor performance," in *Proceedings of the 27th Annual International Symposium on Microarchitecture*, MICRO 27, (New York, NY, USA), p. 22–31, Association for Computing Machinery, 1994.

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

[4] R. Kessler, E. McLellan, and D. Webb, "The alpha 21264 microprocessor architecture," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273)*, pp. 90–95, 1998.

[5] 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.