CSE240A Project

Jiale Xu, Xinhao Luo

TOTAL POINTS

32 / 32

QUESTION 1

1 Report 32 / 32

✓ **- 0 pts Correct**

- 5 pts Literature survey

- 1 pts Bad format: please use formal paper format!

- 3 pts The report is too short and missing too many important implementation details.

- 1 pts Option 2: significantly wrong gshare result

- 1 pts Option 2: significantly wrong tournament result

- 1 pts Option 2: custom predictor does not beat either gshare or tournament

- 1 pts Survey: the structure of the paper is confusing

- 1 pts Survey: too short

- 1 pts Survey: missing detailed elaboration on each method/technology

gradescope

CSE240 Branch Predictor Report

Xinhao Luo, Jiale Xu
University of California, San Diego
San Diego, USA
{x4luo,jix012}@ucsd.edu

1 Introduction

Branch prediction is to make a guess of a branch outcome with information provided without actual execution, and in modern CPU, there is a module responsible for this functionality, branch predictor. Branch predictor is used to accelerate pipeline instruction, avoiding stall when prediction is correct. Without a predictor, the CPU has to wait for the result of the branch to decide which instruction to execute, wasting cycles and bringing delays, causing lower throughput. On the other hand, a predictor is an extra component added to the CPU. It brings overheads to all branch instructions by default, and if the prediction misses, it would incur penalty and potentially cause more time to recover — flushing prefetched instructions, etc. To achieve more accurate predictions, spaces for history patterns are also required. Not only time but space costs are incurred if added. A balance between such overhead and benefits should be carefully considered before moving on with a predictor.

The trend of branch predictors is evolving with the scale of the software these days. From the most simple one like always predicting taken/not taken[1], to Gshare, considering history outcome, and Tournament, considering the difference between local and global outcome. Predictors tend to gather more information, more complex decision processes, to achieve better outcomes. However, limits by the reaction time as well as cost restriction. Predictors are forced to improve their information density — finding better ways of encoding information and keeping those only useful for decision making.

Our paper consists of work from both authors, Xinhao and Jiale. Xinhao has worked on implementations of all predictors mentioned in the paper, as well as the introduction of the paper. Jiale has worked on verifying the implementation, gathering information, providing insights/directions, and building the final observation and conclusion.

2 Implementations

This project has been implemented with three types of branch predictors: Gshare, Tournament, and Custom branch predictor. [1]

[1]We reference some external lecture material during the process and we have listed some here we found useful:
https://people.engr.ncsu.edu/efg/521/f02/common/lectures/notes/lec16.pdf
https://people.cs.pitt.edu/~childers/CS2410/slides/lect-branch-prediction.pdf

2.1 Gshare

Gshare consists of three helper functions `gshare_init()`, `gshare_make_prediction()`, and `gshare_train_predictor()`. The driver program first calls the initialize function to allocate memory for a structure that stores Gshare history table. Then driver calls `gshare_make_prediction()` to return the Gshare prediction. Gshare combines the pc branch address and global branch history to create the index into the Gshare history table by doing an XOR on them.

After this prediction has been returned, the Gshare history table is updated by `update_gshare()`. The driver provides the input of branch address and a outcome signal indicating branch instruction being taken or not. `update_gshare()` takes these two inputs and computes the two-bit predictor by calling `update_two_bit_predictor()`. The current two-bit predictor is one of the four possible status of `00, 01, 10, 11`. Regarding these four status, the predictor returns TAKEN if $status > 01$, else returns `NOTTAKEN`. Gshare integrates the current outcome into the two-bit predictor by the following mapping. If outcome is `NOTTAKEN`, `00` stays as `00`; update `01` to `00`; update `10` to `01`; update `11` to `10`. If outcome is `TAKEN`, `11` stays as `11`; update `10` to `11`; update `01` to `10`; update `00` to `01`. This completes the procedure to update the Sshare global branch history table. Gshare uses a dictionary with branch address as index and predictor as value. In this project implementation, Gshare uses 13 Gshare history bit, so Gshare program has 2^{13} branch addresses, and for each branch address, it has two bit to store its predictor. Thus, the total memory cost for Gshare is computed as $2^{13} \times 2 = 2^{14} = 16384$ bits with a rounding of 16 Kbits.

2.2 Tournament

Tournament branch predictor is implemented with `tournament_init()`, `tournament_make_prediction()`, and `update_tournament()`, three main helper functions. The procedure is generally similar to Gshare, but their structures and implementations differ from each other. Tournament has three parameters that is determined at the start of the program:

- pcIndexBits
- lhistoryBits
- ghistoryBits

The driver program calls `tournament_init()` to allocate memory for a local history table with the size of $2^{pcIndexBits} \times lhistoryBits$, local predictors with the size of $2^{lhistoryBits} \times 2$, global predictors with the size of $2^{ghistoryBits}$, and choice

predictors with the size of $2^{ghistoryBits}$. In the project implementation, driver program uses the setting `9:10:10` for `ghistoryBits:lhistoryBits:pcIndexBits`. So the overall space complexity for the whole Tournament branch predictor is $2^{10} \times 10 + 2^{10} \times 2 + 2^9 \times 2 + 2^9 \times 2 = 14336$ bits with a rounding of 14 Kbits. `tournament_make_prediction()` has two helper functions that returns the global branch prediction and local branch prediction. After the memory allocation, the driver calls `tournament_make_prediction()` to read the current choice predictor and decide to take the local prediction or global prediction. `update_tournament()` calls two helper functions to update the choice predictors, and update the local branch predictors and global branch predictors. This completes the implementation of the tournament branch predictor.

Comparing to Gshare branch predictor, Tournament has two more map structures that not only store local global branch predictors, but also store the outcome predictors that indicate the correctness of previous prediction. Consider the project's setting of `gshare:13` and `tournament 9:10:10`, Gshare allocates comparably more memory than tournament with Gshare using 13-bit branch addresses as index to the branch history table. Tournament, on the other hand, captures 9-bit global branch history and 10-bit local branch history.

2.3 Custom

The project aims to build a Custom predictor with space usage constraint and want to achieve better prediction success rates than the previous two branch predictors. The maximum memory usage of Custom branch predictor is set to $64Kbits + 256bits$. The other initiative is to beat both Gshare's and Tournament's prediction success rates in all 6 traces of branch instructions. Our team decides to build a Custom predictor that decide from both Gshare and Tournament prediction based on the outcome predictor as the first attempt to approach this initiative.

Generally speaking, Custom predictor is a revised kind of Tournament predictor. The Custom predictor revises the global prediction and choice prediction of the Tournament predictor. Custom's global prediction is replaced by Gshare prediction structure. And the choice prediction is switched from two-bit predictor to three-bit predictor. Similar to the Tournament predictor, the Custom predictor is implemented with `custom_init()`, `custom_make_prediction()`, and `update_custom()`. In `init_predictor()`, if branch predictor is Custom type, the three variables, denote as `g:l:p` (ghistoryBits, lhistoryBits, and pcIndexBits) are set to fixed values to build Gshare and Tournament predictors. This Custom predictor has the `g:l:p` value as `13:13:10`. After the local variable initialization, driver calls `custom_make_prediction()` to return the prediction following the outcome predictor which stores three-bit choice history. The outcome predictor chooses from the result of two subroutines:

- `gshare_make_prediction()`
- `tournament_local_prediction()`

Then, driver calls `update_custom()` to update both Gshare and Tournament variables. `update_custom()` is similar to `update_tournament()` except that the `tournament_update_choice()` is replaced with `update_gshare()`, and the choice predictor is three-bit.

3 Observation

For custom predictor, the strict constraint of maximum memory usage is the first challenge. Note that the global branch history is replaced with the Gshare history table with the length of `ghistoryBits`. Also, the two-bit predictor is switched to a three-bit predictor which increases the outcome predictors total memory size, so the formula for the custom predictor's total memory usage is as equation 1.

$$\begin{aligned} total\_memory &= local\_history\_table\_memory \\ &\quad + local\_prediction\_memory \\ &\quad + gshare\_prediction\_memory \\ &\quad + outcome\_prediction\_memory \\ &= 2^{pcIndexBits} \times lhistoryBits \\ &\quad + 2^{lhistoryBits} \times 2 + 2^{ghistoryBits} \times 2 \\ &\quad + 2^{ghistoryBits} \times 3 \end{aligned} \tag{1}$$

Here we are trying to find set of parameters with total memory usage below 64 Kbits while yielding best results. We assume that branch history table with longer branch addresses indices generates better prediction results than history table with shorter branch address. We want to study how MR correlates with the length of `ghistoryBits` by running test on three custom predictors with `lhistoryBits` and `pcIndexBits` fixed. Testing results for custom predictor C1(9:10:10), C2(10:10:10), and C3(11:10:10) is computed as the figure below. Figure 1 gives a direct comparison among Gshare, Tournament and three custom predictors. We can see that three custom predictors beat Gshare and Tournament with traces int2, mm1, and mm2.

Figure 1 also indicates that with `lhistoryBits` and `pcIndexBits` fixed, increases in `ghistoryBits` lead to increase in MR regarding test results of traces fp2 and int1. Also increases in `ghistoryBits` do not affect the test results of trace fp1. According to the memory calculation, the total memory usage requirement is met for each of custom predictors. Among these three custom predictors, C3 gives better results than the other two custom predictors. Following the memory calculation formula for custom predictor, the space usage for C3 is computed as 32768 bits, which is less than 64 Kbits requirement.

trace\predictor	GS	T	C1	C2	C3
fp1	0.825	0.991	0.994	0.995	0.997
fp2	1.678	3.246	0.777	2.069	2.069
int1	13.839	12.622	11.226	11.698	12.370
int2	0.420	0.426	0.300	0.307	0.316
mm1	6.696	2.581	1.364	1.421	1.822
mm2	10.138	8.483	7.455	7.936	8.585

Figure 1. Two-bit Outcome Predictor

Then we switch to the three-bit outcome predictor with the same `g:l:p` setting of C1, C2, and C3. Figure 2 shows the MR results which indicates that switching from two-bit choice predictor to three-bit choice predictor produces better MR results.

trace\predictor	GS	T	C1	C2	C3
fp1	0.825	0.991	0.985	0.985	0.985
fp2	1.678	3.246	2.253	2.253	1.514
int1	13.839	12.622	13.343	12.507	11.510
int2	0.420	0.426	0.375	0.341	0.321
mm1	6.696	2.581	2.422	1.934	1.745
mm2	10.138	8.483	8.515	8.330	8.031

Figure 2. Three-bit Outcome Predictor

The six branch instruction traces form 12 test cases regarding beating each of the two branch predictor Gshare and Tournament. Up to this point, Custom predictor C3 successes in beating 11 cases out of the 12 test cases. And the memory usage requirement leaves a size of $64 - 32 = 32$ Kbits space usage. Following the correlation observed from Figure 1 and 2, the straightforward study direction is to increase `ghistoryBits` to 13, and increases other two variables as large as permitted by the space usage requirement. By considering a set of possible settings with `ghistoryBits` fixed as 13, and both `lhistoryBits` and `pcIndexBits` in the range $(8, 15)$, the total memory is calculated by the memory usage requirement. Sorting the memory usage of each `g:l:p` of this set, the setting of `13:13:10` has the largest memory usage which is exactly 64 Kbits.

The results for each of the six address traces, and for each of the Gshare(GS), Tournament(T), and Custom(C_pick:13:13:10) branch predictors are in Figure 3

trace\predictor	GS	T	C_pick
fp1	0.825	0.991	0.814
fp2	1.678	3.246	0.960
int1	13.839	12.622	9.722
int2	0.420	0.426	0.275
mm1	6.696	2.581	0.559
mm2	10.138	8.483	6.875

Figure 3. 13:13:10 Outcome Predictor

Although `C_pick` has performed well regarding to its MR result, the space consumption is pretty large. The total memory usage for `C_pick` is approximately 4 times the size of Gshare:13 and Tournament:9:10:10. Trading between space storage and prediction performance is the main concept when designing the Custom predictor. The `g:l:p` setting contains three variables that describes the length of three local variables in the Custom predictor program, which directly gives three directions that might be correlated with MR. We have analyzed the correlation between MR and ghistoryBits length. And the `lhistoryBits` and `pcIndexBits` might also give some correlation with MR. Given our observation that $ghistoryBits = 13$ produces better MR results, we want to fix `ghistoryBits` to 13 and study the relationship of MR with `lhistoryBits` and `pcIndexBits`. First we study the correlation between MR and `lhistoryBits`, build four predictors with $ghistoryBits = 13$, and other variables set as follows:

- bp1(13:9:10)
- bp2(13:10:10)
- bp3(13:11:10)
- bp4(13:12:10)

trace\predictor	bp1	bp2	bp3	bp4
fp1	0.813	0.813	0.814	0.815
fp2	1.121	1.123	1.123	0.960
int1	9.894	9.847	9.794	9.757
int2	0.296	0.289	0.288	0.280
mm1	2.180	1.416	1.133	0.672
mm2	7.192	7.108	7.055	6.960

Figure 4. lHistoryBits Correlation

Based on Figure 4, we find that these four branch predictors generally has better MR results with `lhistoryBits` increasing, but the MR improvement is relatively small. We started from consuming up the maximum memory usage restriction of 64 Kbits and ended up with 13:13:10 setting. At the same time, we find parameter 13:9:10 also passes the 12 test case, that is, beats both Gshare and Tournament with all six traces. And we want to achieve some memory space efficiency by reducing `lhistoryBits`. Thus, we choose 13:9:x as the final setting for the final Custom predictor design regarding `ghistoryBits` and `lhistoryBits`. Then, we analyze the correlation between MR and `pcIndexBits`, build four predictors with $ghistoryBits = 13$, and other variables set as follows:

- bp5(13:9:9)
- bp6(13:9:10)
- bp7(13:9:11)
- bp8(13:9:12)

trace\predictor	bp5	bp6	bp7	bp8
fp1	0.818	0.813	0.811	0.811
fp2	1.122	1.121	0.383	0.383
int1	10.281	9.894	9.675	9.675
int2	0.304	0.296	0.289	0.289
mm1	2.602	2.180	2.225	2.225
mm2	7.690	7.192	6.819	6.819

Figure 5. pcIndexBit Correlation

Based on Figure 5, we find that these four branch predictors generally has better MR results with pcIndexBits increasing, but the MR improvement stops at $pcIndexBits = 11$. Following the memory usage formula, bp5, bp6, and bp7 memory usages are calculated to be 46592, 51200, and 60416 bits. In order to achieve the best performance with our current Custom predictor among all possible settings, we set our final setting to be 13:9:11 to achieve the MR results as the bp7 shown above.

4 Result and Conclusion

Following from observations above, we define Custom predictor with 13:9:11 as the `g:l:p` setting and achieved the final MR results shown in Figure 6.

trace\predictor	GS	T	C_pick
fp1	0.825	0.991	0.811
fp2	1.678	3.246	0.383
int1	13.839	12.622	9.675
int2	0.420	0.426	0.289
mm1	6.696	2.581	2.225
mm2	10.138	8.483	6.819

Figure 6. Final Result

The Custom predictor originates from the Tournament predictor and developed through several improvements. Based on figure 1 and 2, we obtained the observation that C1, C2, and C3 predictors have better MR results using three-bit outcome predictor than using two-bit outcome predictor. Then we tested MR results regarding different `ghistoryBits`, and observed that $ghistoryBits = 13$ gives better results. By consuming up the maximum memory usage, we set `ghistoryBits :lhistoryBits:pcIndexBits` = 13:13:10. Later, we conducted two set of testings to find out MR's correction with length of `lhistoryBits` and `pcIndexBits`.

By comparing results among all Custom branch predictors, we finalized the Custom predictor to have the setting of 13:9:11 with 60416 bits memory usage regarding the predictor structures. And the local numerical variables memory usage is below the 256 bits requirement. Also, the final Custom predictor beats both Gshare and Tournament predictors regarding each of the six traces.

References

[1] James E Smith. 1998. A study of branch prediction strategies. In *25 years of the international symposia on Computer architecture (selected papers)*. 202–215.

1 Report 32 / 32

✓ **- 0 pts** **Correct**

- 5 pts Literature survey

- 1 pts Bad format: please use formal paper format!

- 3 pts The report is too short and missing too many important implementation details.

- 1 pts Option 2: significantly wrong gshare result

- 1 pts Option 2: significantly wrong tournament result

- 1 pts Option 2: custom predictor does not beat either gshare or tournament

- 1 pts Survey: the structure of the paper is confusing

- 1 pts Survey: too short

- 1 pts Survey: missing detailed elaboration on each method/technology

gradescope