NC State University

Department of Electrical and Computer Engineering

ECE 463/563: Fall 2021 (Rotenberg)

Project #2: Branch Prediction

by

<< Seebani Mahapatra >>

NCSU Honor Pledge: "I have neither given nor received unauthorized aid on this project."

Student's electronic signature: ____Seebani Mahapatra_____
(sign by typing your name)

Course number: _____563_____
(463 or 563 ?)

BRANCH PREDICTORS

The branch predictor role is to improve the flow in the instruction pipeline. Branch predictors play a critical role in achieving high effective performance in many modern pipelined microprocessor architecture.

The branch predictor is a digital circuit that guesses the outcome of a conditional operation and showcases the most likely result. It speeds up the processing of branches in the CPU using pipelines.

Before Branch predictors were introduced, the processor would wait until the conditional jump 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 whether the conditional jump is most likely to be taken or not taken.

But if the guess is incorrect, the instructions fetched or partially executed will be dumped and it starts over with the correct outcome resulting in delays. These incorrect guesses are otherwise known as branch mispredictions.

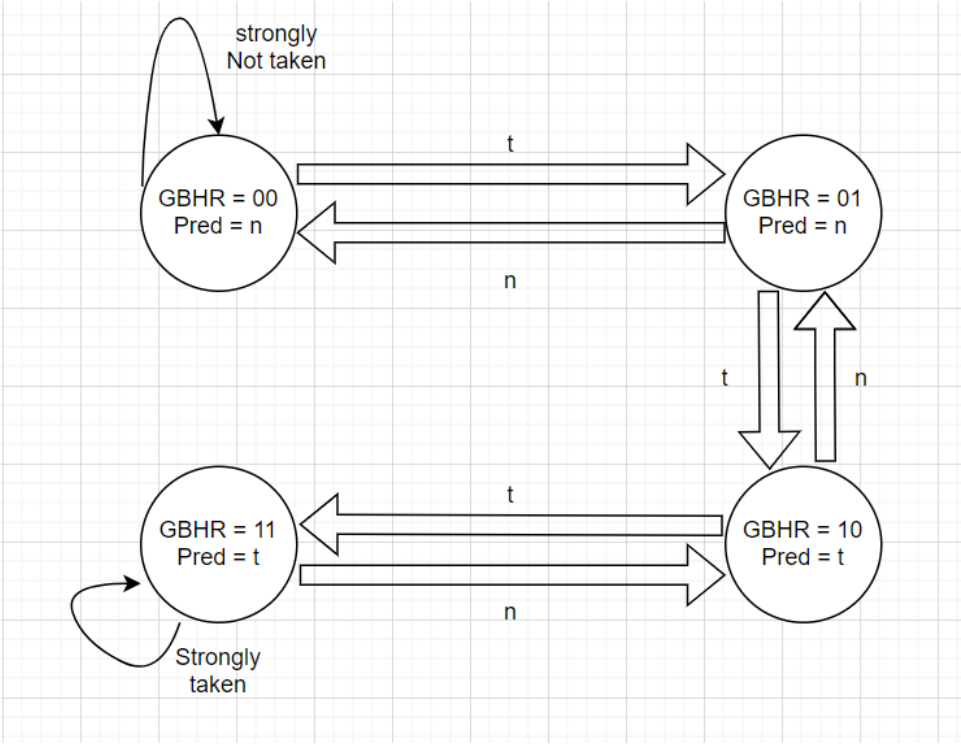
Logic of branch predictor: A prediction is made for a branch currently residing in the branch and will be decided if the prediction is taken or not depending upon its validity. If the prediction is true, then the pipeline is not squashed, and clock cycles are not lost. But if the prediction turns out to be false then the pipelines are squashed, and clock cycles are lost as it starts over with the current circumstance.

A branch target buffer is the table that keeps an account of destination of the branch and history of the branch whether it was taken or not. The BTB acts as a cache that is located near the decode stage and it monitors the branch instruction.
The first time the branch instruction enters the pipeline, BTB looks into itself about its history. But since it never entered before, it will be a BTB miss and predicts that the branch will not be taken.

On reaching the execution stage, it is decided if the branch will be taken or not taken. If it is taken, the next instruction will be fetched from the branch target address, else the next instruction will be sequentially fetched.

When the branch is taken for the first time, the execution stage sends feedback to the prediction table and the address sent back is recorded in BTB.

FSM for branch prediction:

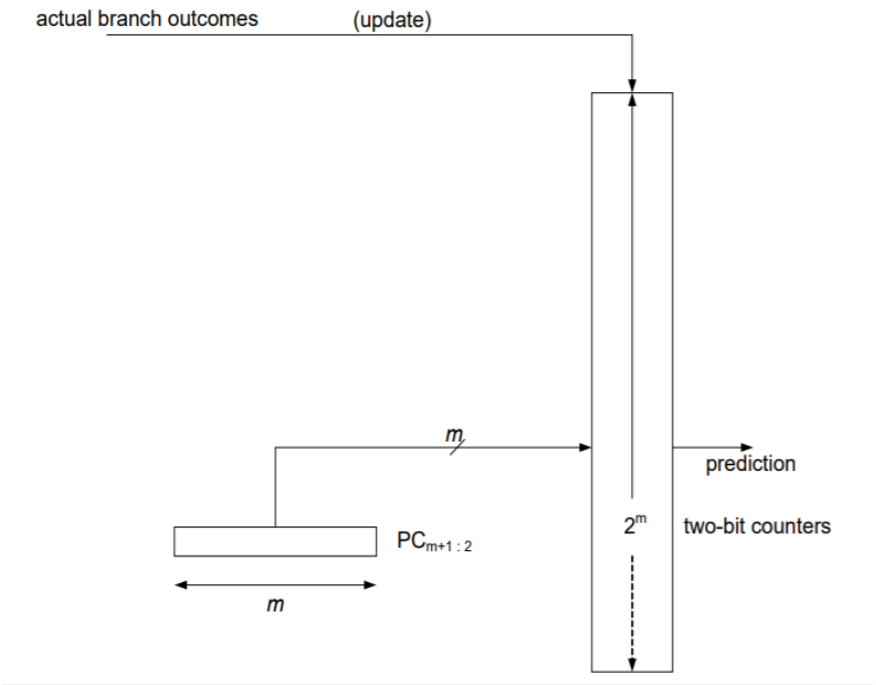


‘t’ = taken, ‘n’ = not taken, GBHR = global branch history register

The GBHR is a two-bit counter. Whenever the branch is taken, it increments but saturates at 3. So, if the branch is taken at the next instruction, it remains in the same state. But if the branch is not taken it decrements and saturates at 0. So, even if the next branch is not taken for the next instruction, it doesn't decrement further and remains in the same state. Hence, state 0 and state 3 are considered as strongly not taken and strongly taken respectively. The other states are known as, state 1 = weakly not taken and state 2 = weakly taken.

There are many branch predictors to make these guesses. This project mainly focuses on bimodal branch predictor, gshare branch predictor and hybrid (bimodal + gshare) branch predictor.

Bimodal Branch predictor:

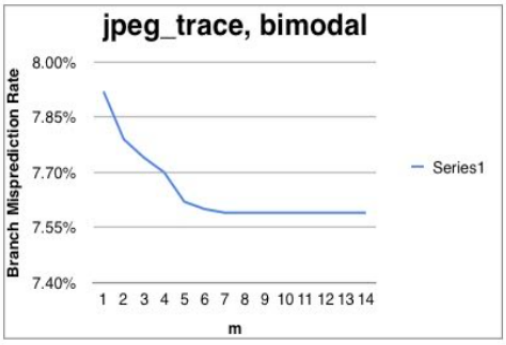


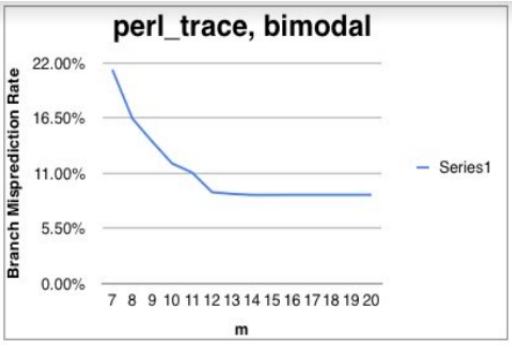
Algorithm:

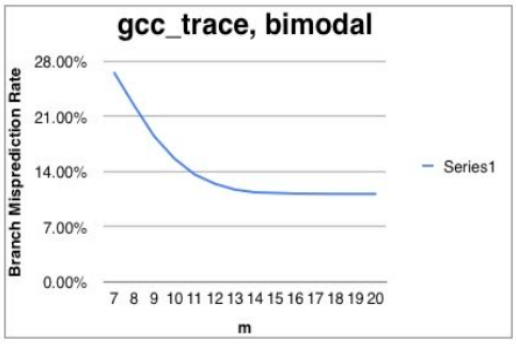
After receiving the branch from the trace file, there are three steps:
(1) Determine the branch's index into the prediction table.
(2) Make a prediction. Use index to get the branch's counter from the prediction table. If the counter value is greater than or equal to 2, then the branch is predicted taken, else it is predicted not taken.

(3) Update the branch predictor based on the branch's actual outcome. The branch's counter in the prediction table is incremented if the branch was taken, decremented if the branch was not taken. The counter saturates at extremes (0 and 3), however.

Bimodal Outcome:





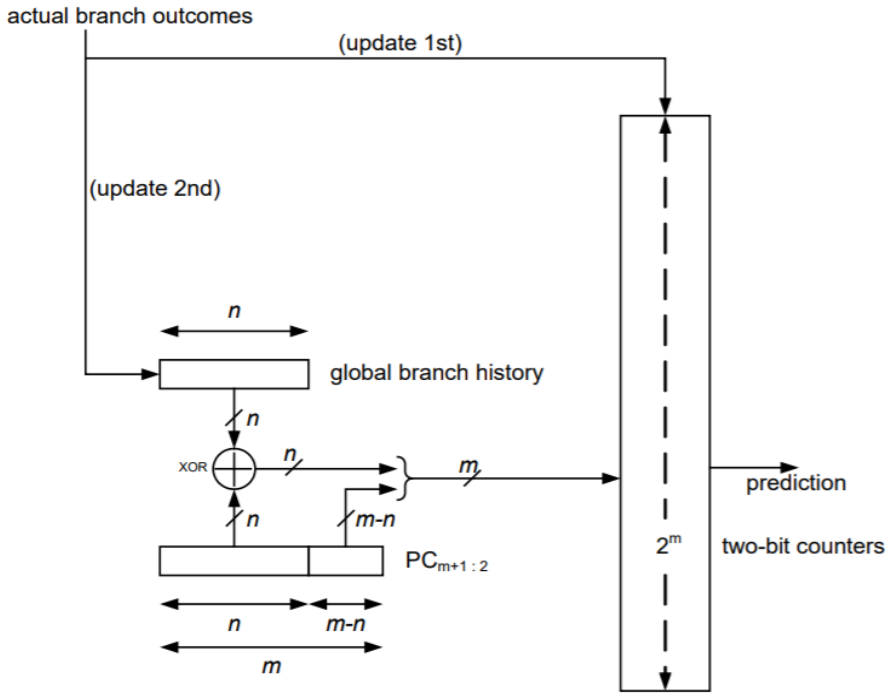


The simplest dynamic branch direction predictor is an array of 2n two-bit saturating counters. Each counter includes one of four values: strongly taken (T), weakly taken (t), weakly not taken (n), and strongly not taken (N).

Prediction- To make a prediction, the predictor selects a counter from the table using the lower-order m bits of PC (Program Counter). The direction prediction is made based on the value of the counter.

Training. - After each branch (correctly predicted or not), the hardware increments or decrements the corresponding counter to bias the counter toward the actual branch outcome (the outcome given in the trace file). As these are two bit saturating counters, decrementing a minimum counter or incrementing a maxed-out counter should have no impact.
The branches are data dependent (let's say the data is randomly distributed), so the bimodal is not likely to capture them. So, if the second data depends on the first data the bimodal will be insufficient to capture that.
This can be overcome using Gshare as it uses global predictor.

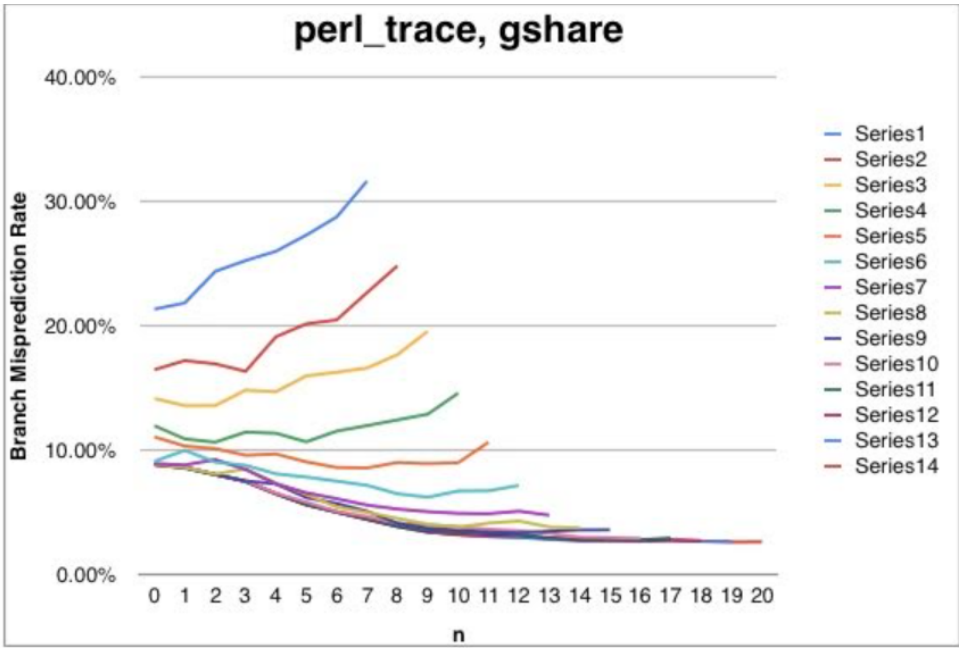
Gshare Branch predictor:

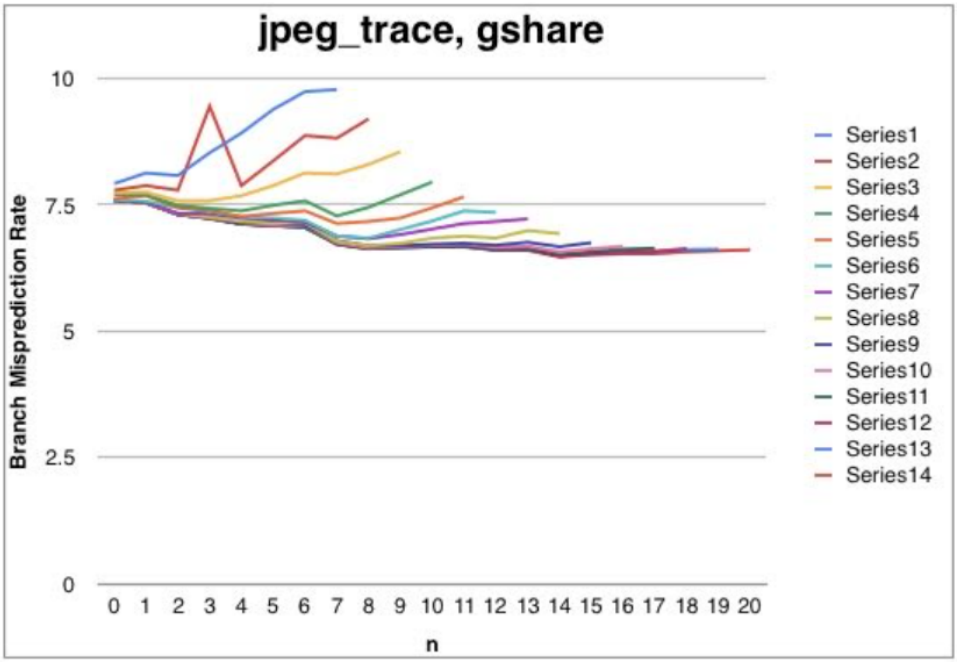


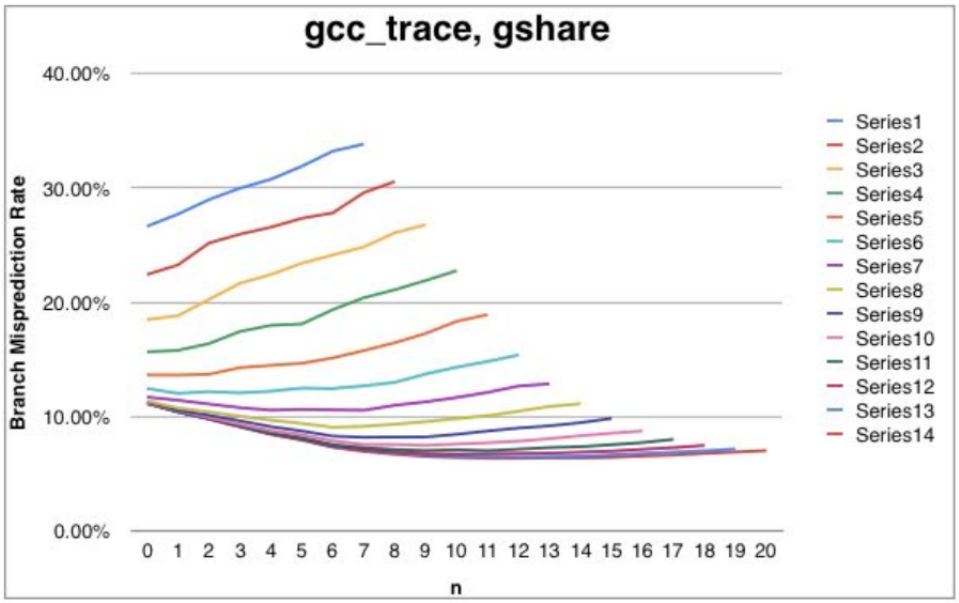
Algorithm:
After receiving the branch from the trace file there are four steps:
(1) Determine the branch's index into the prediction table. The fig. Above depicts how to generate the index: the current n-bit global branch history register is XORed with the uppermost n bits of the m PC bits.

(2) Make a prediction. Use index to get the branch's counter from the prediction table. If the counter value is greater than or equal to 2, then the branch is predicted taken, else it is predicted not taken.
(3) Update the branch predictor based on the branch's actual outcome. The branch's counter in the prediction table is incremented if the branch was taken, decremented if the branch was not taken. The counter saturates at extremes (0 and 3), however.
(4) Update the global branch history register. Shift the register right by 1 bit position and place the branch's actual outcome into the most-significant bit position of the register.

Gshare outcome:







The Gshare “global sharing” branch predictor is a (2, 2) predictor. It uses both global branch history and the location of a branch instruction to create an index into a table of 2-bit counters. The Gshare architecture uses an m-bit global history register to keep track of the direction of the last m executed branches. To simplify the implementation, this global

history register is xored with the (PC>>2) to create an index into a 2m-entry pattern history table of n-bit counters. The result of this index is the prediction for the current branch. The predictor then compares this prediction with the real branch direction to determine if the branch was correctly predicted or not, and updates the prediction statistics. The predictor then updates the n-bit counter used to perform the prediction. The counter is:

- incremented if the branch was taken
- decremented if the branch was not taken.

Finally, the branch outcome is shifted into the most significant bit of the global history register.

Hybrid Branch Predictor:

Algorithm:

After receiving the branch from the trace file, follow these steps:

(4) Update the selected branch predictor based on the branch's actual outcome. Only the branch predictor that was selected in step 3, above, is updated (if gshare was selected, follow step 3 in gshare otherwise follow step 3 in bimodal).
(5) The gshare's global branch history register must always be updated, even if bimodal was selected.
(6) Update the branch's chooser counter using the following rule:

	Results from predictors:		
	both incorrect or both correct	***gshare correct, bimodal incorrect***	***bimodal correct, gshare incorrect***
Chooser counter update policy:	no change	increment (but saturates at 3)	decrement (but saturates at 0)

Conclusion:

The bimodal is likely to learn faster and have less collisions, but the global variants are more elaborate and have better chance to capture complicated patterns. However, it has higher collisions since the tables represent all sorts of partial histories, and has lower chances to converge in some cases.

- Bimodal - use when local history is beneficial
- Gshare –use when global history is beneficial

Therefore, we can use hybrid of both the predictors to achieve maximum efficiency.