NC State University

Department of Electrical and Computer Engineering

ECE 463/563: Fall 2020 (Rotenberg)

Project #2: Branch Prediction

by

Evan Mason

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

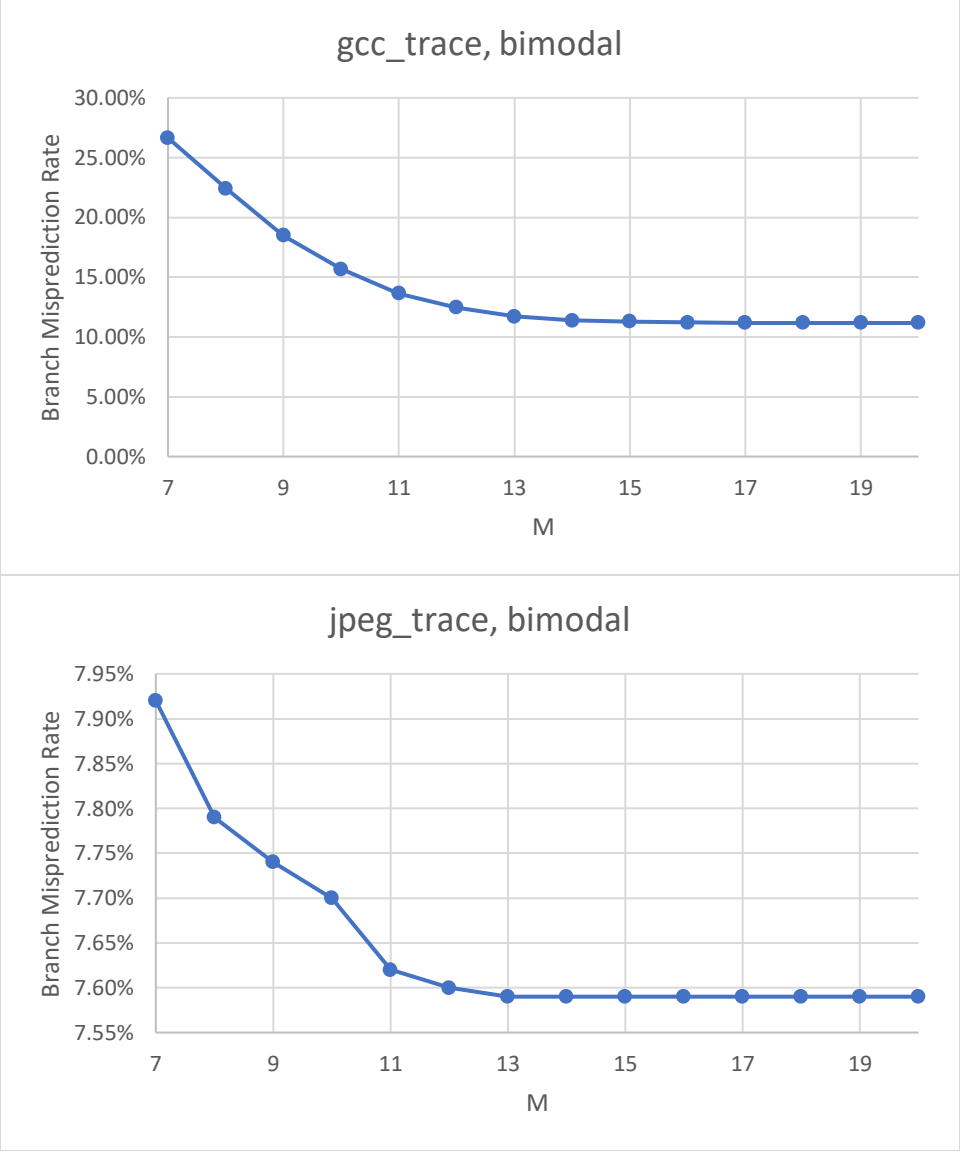
Student's electronic signature: Evan Mason
(sign by typing your name)

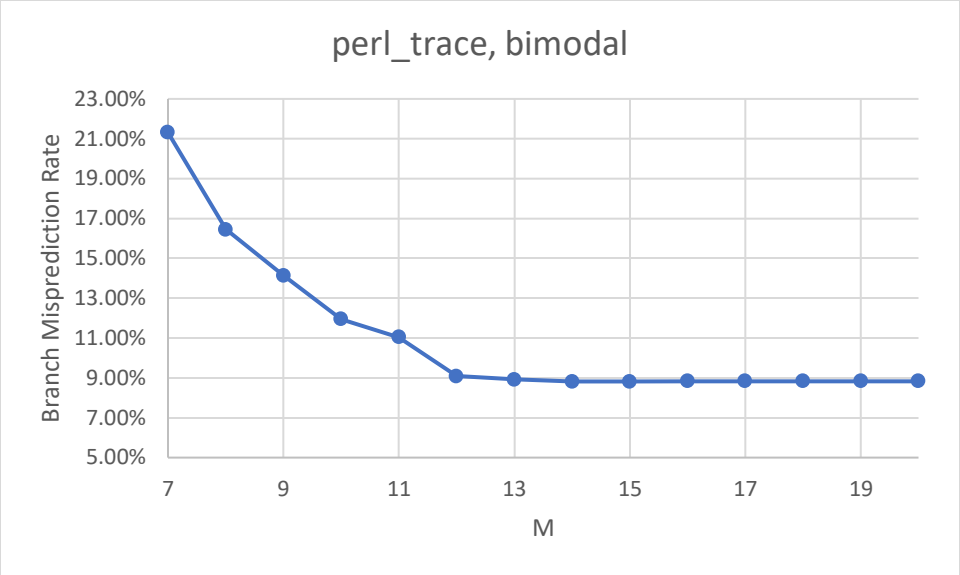
Course number: **_463**

PART 1: BIMODAL PREDICTOR

(a) [ECE463: 25 points] or [ECE563: 20 points] Match the four validation runs "val_bimodal_1.txt", "val_bimodal_2.txt", "val_bimodal_3.txt", and "val_bimodal_4.txt", posted on the website for the BIMODAL PREDICTOR. Each validation run is worth ¼ of the points for this part (6 or 5 points each). You must match all validation runs to get credit for the experiments with the bimodal predictor, however.

(b) [ECE463: 25 points] or [ECE563: 20 points] Simulate BIMODAL PREDICTOR for different sizes ($7 \leq m \leq 20$). Use the traces gcc, jpeg, and perl in the trace directory. [20 or 15 points] Graphs: Produce one graph for each benchmark. Graph title: "<benchmark>, bimodal". Y-axis: branch misprediction rate. X-axis: m. Per graph, there should be only one curve consisting of 14 datapoints (connect the datapoints with a line).





[5 points] Analysis: Draw conclusions and discuss trends. Discuss similarities/differences among benchmarks.

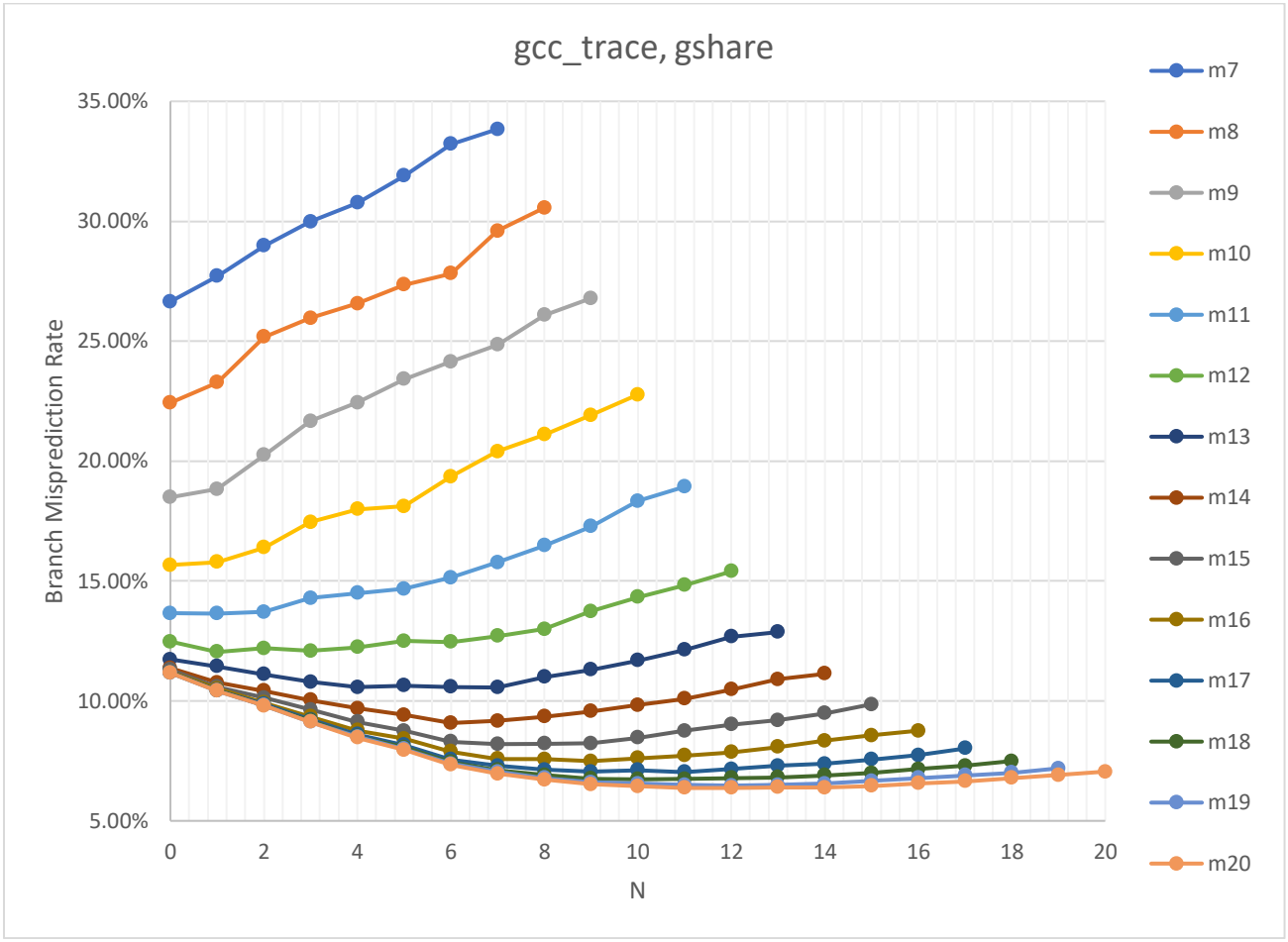
It's pretty apparent that as the number of low-order PC bits (M) increases, the overall branch misprediction rate for a bimodal branch predictor decreases. All three relatively follow a negatively exponential path. Some things to point out are the subtle change in the amount that branch misprediction rate decreases for the jpeg and perl benchmarks. Specifically, when M = 8, 9, and 10 the bimodal branch predictor isn't as effective in terms of the amount of improvement when increasing the size of M. Although all three benchmarks have relatively the same shape of improvement, the bimodal branch predictor preformed best on the jpeg trace, perhaps the same branch instructions were frequently called. All benchmarks eventually stall out roughly when M=13.

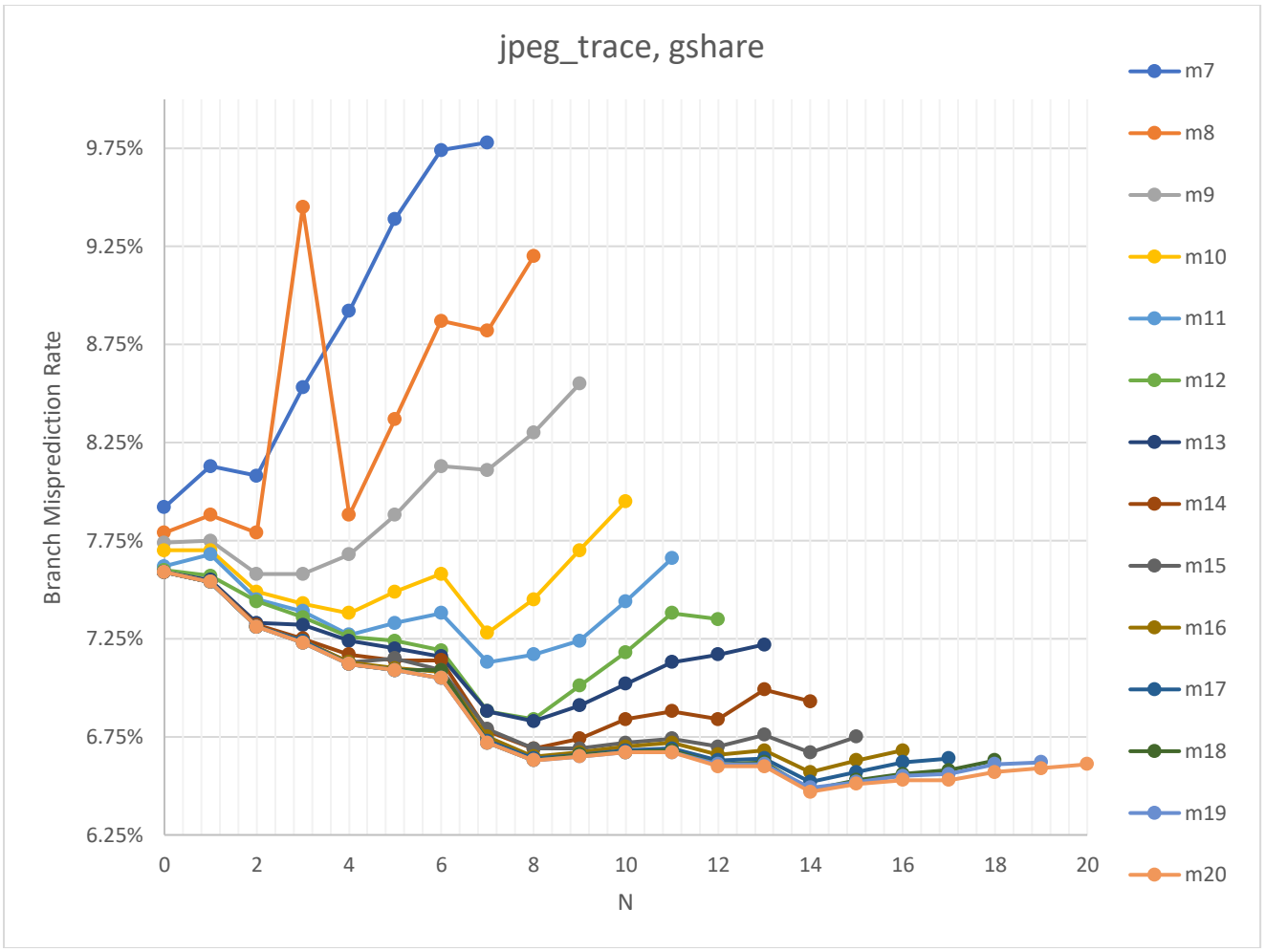
PART 2: GSHARE PREDICTOR

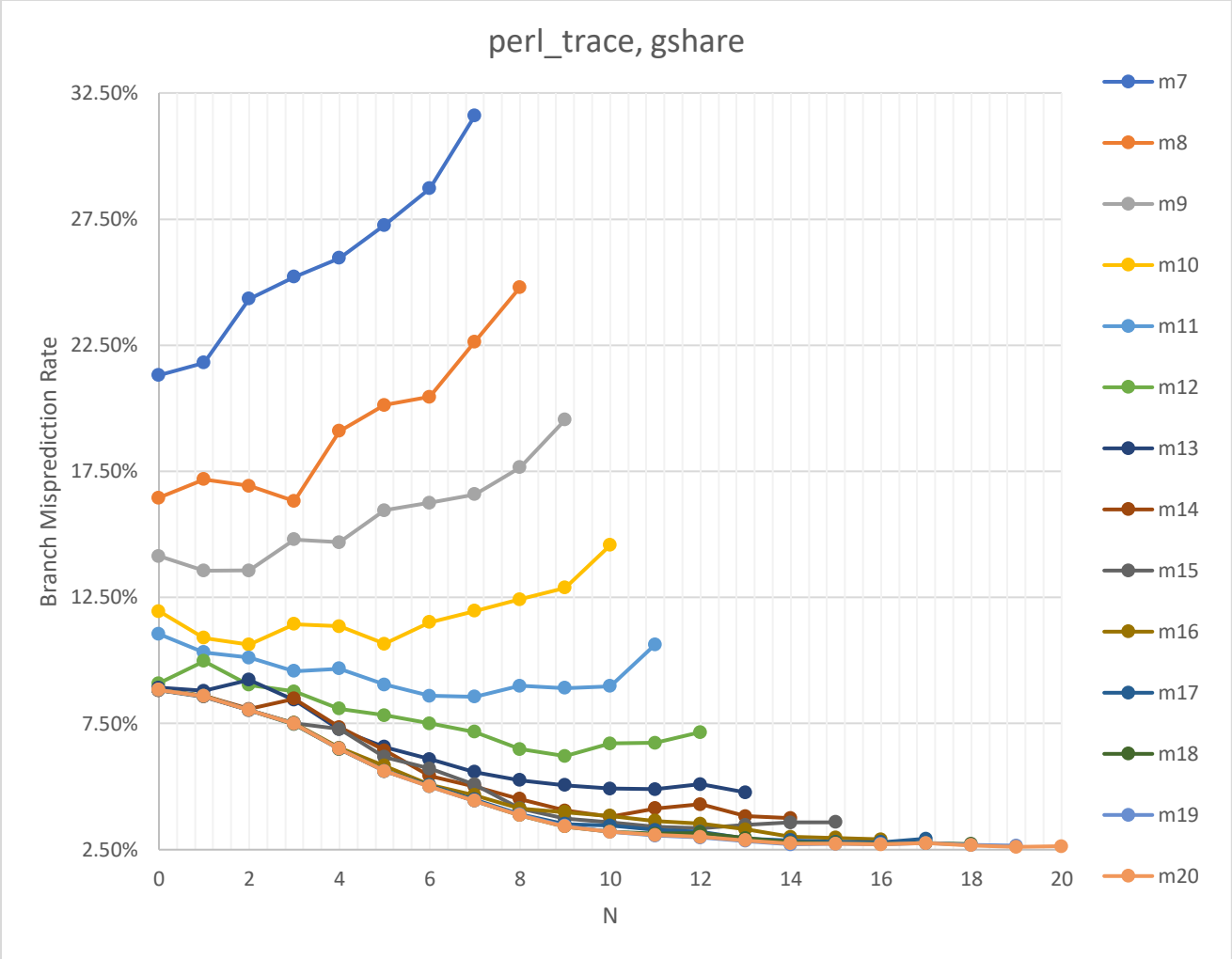
(a) [ECE463: 25 points] or [ECE563: 20 points] Match the four validation runs "val_gshare_1.txt", "val_gshare_2.txt", "val_gshare_3.txt", and "val_gshare_4.txt", posted on the website for the GSHARE PREDICTOR. Each validation run is worth ¼ of the points for this part (6 or 5 points each). You must match all validation runs to get credit for the experiments with the gshare predictor, however.

(b) [ECE463: 25 points] or [ECE563: 20 points] Simulate GSHARE PREDICTOR for different sizes ($7 \leq m \leq 20$), and for each size, i.e., for each value of m, sweep the global history length n from 0 to m. Use the traces gcc, jpeg, and perl in the trace directory. [20 or 15 points] Graphs: Produce one graph for each benchmark. Graph title: "<benchmark>, gshare". Y-axis: branch misprediction rate. X-axis: n (spanning n=0 to n=20). Per graph, there should be a total of 203 datapoints plotted as 14 curves. Datapoints having the same value of m (same predictor size) are connected with a line, i.e., one curve for each value of m. Note that not all curves have the same number of datapoints; see the listing below for the number of datapoints for each of the 14 curves,

m=7 through m=20. The rationale for this graph is to study the effect of global history length for each predictor size.




jpeg_trace, gshare
Branch Misprediction Rate
9.75%
9.25%
8.75%
8.25%
7.75%
7.25%
6.75%
6.25%
0
2
4
6
8
10
12
14
16
18
20
N
m7
m8
m9
m10
m11
m12
m13
m14
m15
m16
m17
m18
m19
m20



[5 points] Analysis: Draw conclusions and discuss trends. Discuss similarities/differences among benchmarks.

What's interesting about utilizing a global branch history register (BHR), is that there is an inflection point for each graph in terms of the branch misprediction rate. As seen before with a bimodal branch predictor, increasing M (low-order bits of PC instruction) improves/decreases the branch miss prediction rate for a given benchmark. However, after utilizing a global BHR, when M is less than roughly a bit size of 13 the BHR actually hinders the prediction system. As in the overall branch miss prediction rate increases with increasing N (# bits in global BHR), for a simulation with M < 12. As M increases past 12-bits, we can see that utilizing the BHR actually helps the predictor in terms of decreasing the overall branch miss prediction rate. This is only strongly apparent in the perl benchmark, for the gcc and jpeg benchmark it is less apparent. I would still assume the jpeg benchmark consists of frequent requests of the same PC branch instructions. Gcc is likely more frequently different PC branch requests, and the perl benchmark likely toggles. In conclusion, utilizing a global BHR seems to be most effective when M is sufficiently large (M>11).