NC State University

Department of Electrical and Computer Engineering

ECE 463/563: Fall 2019 (Dr. Huiyang Zhou)

Project #2: Branch Prediction

By

Rhushikesh Anand Prabhune

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

Student's Electronic Signature: Rhushikesh Anand Prabhune
(Sign by typing your name)

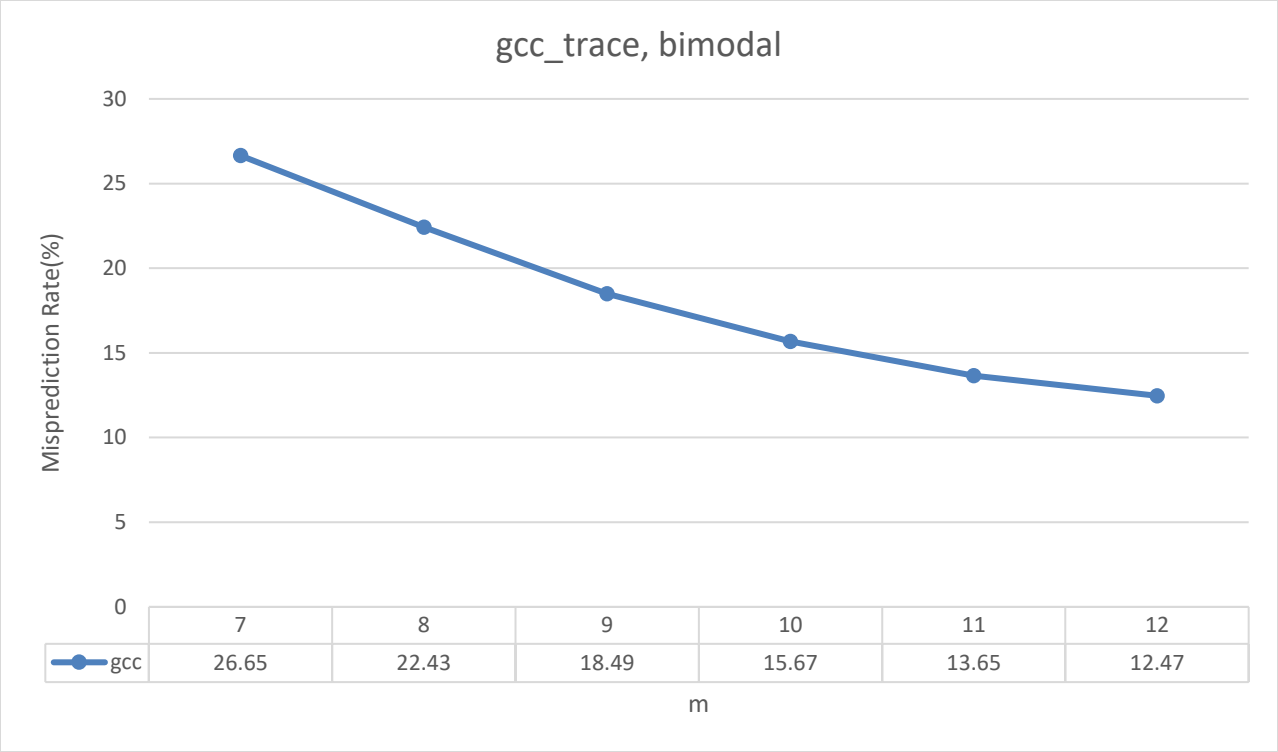
Course Number: 563
(463 or 563?)

Bimodal Predictor:

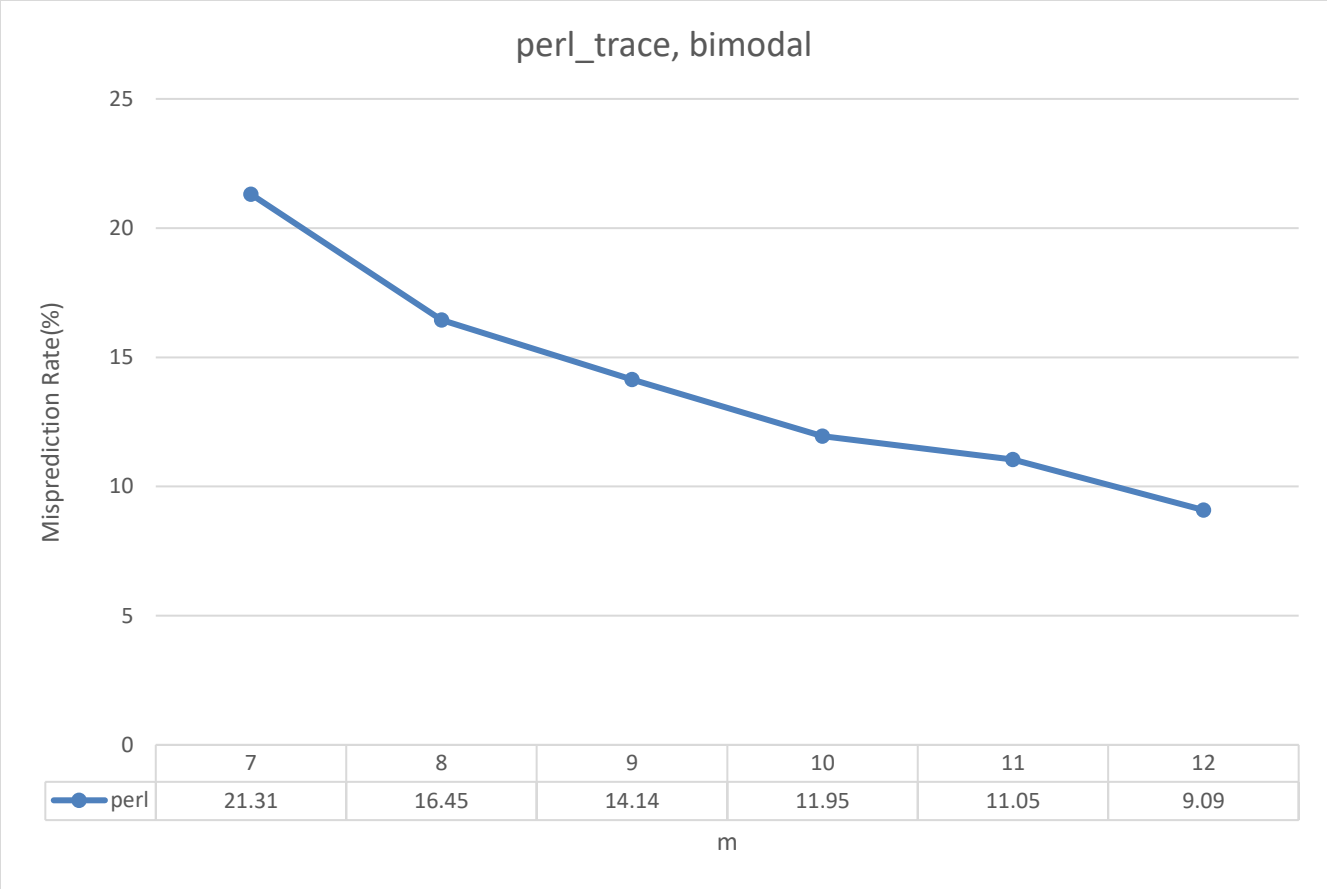
We see that in all the benchmarks, as the number of bits (m) used to index the program counter (PC) increases, the misprediction rate also drops substantially. This is because the number of entries in the prediction table depends on the value of m. Therefore, if m bits are used to index the PC, the number of entries in the prediction table equals two raised to the power of m. More the number of entries, more the number of Smith Counters which correspond to a particular index. Therefore, this index will hold more information about the instruction than an index with lesser number of bits. This will lead to a decrease in interference which will in turn lead to decrease in mispredictions.

For example, a small value of m will result in the same index for multiple branches. Hence, the counter will be updated whenever one of the branches arrive at the predictor which is not always undesirable but can lead to more number of mispredictions. Whereas with a larger index, the counter will be updated fewer number of times because now lesser number of branches correspond to a particular branch. Increasing the size of the index bits will increase the number of Smith counter which will add to the hardware cost. Therefore, an optimal number of bits should be used for the index after examining the misprediction rates and the hardware cost.

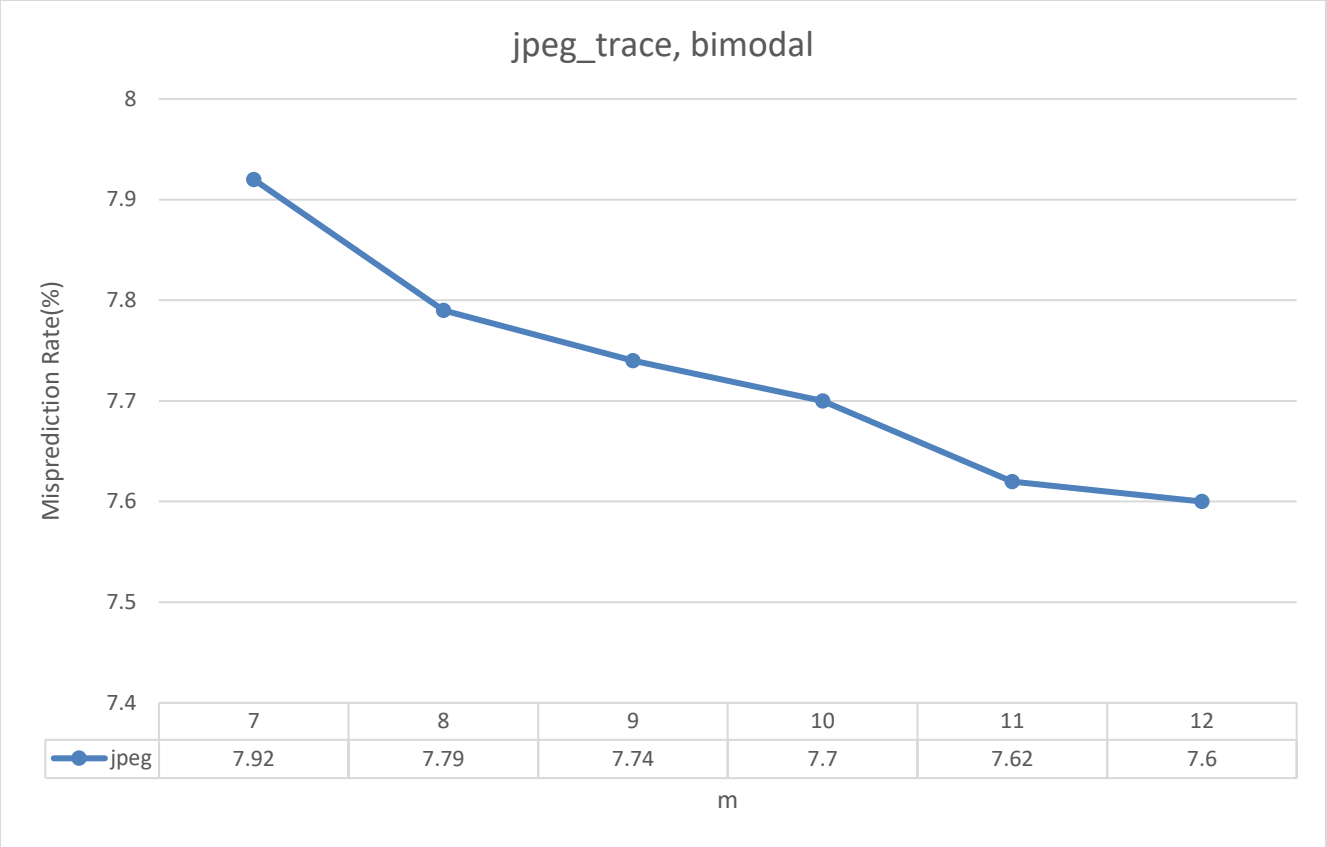
A bimodal branch predictor gives about 70-80% correct predictions. This needs to be a bigger number considering the pipeline bubbles that result from these predictions. In the case where predictions are not made, the pipeline does not suffer from any bubbles if the branch is not taken. However, when a wrong prediction is made the pipeline will suffer from a bubble no matter the branch was taken or not. The bimodal branch predictor can also give a very high misprediction rate in case of some repeating branch patterns. To correct this error, a Branch History Register (BHR) is used which tracks the history of incoming branches. Such a BHR is implemented in the gshare branch predictor in later sections. The graphs plotted from the simulation of a bimodal predictor for three different benchmarks are given below.



The gcc_trace has the highest number of mispredictions for a particular m out of all the benchmarks. A misprediction rate of 26.65 is considerable and this benchmark needs to use a different predictor or a bigger m to get satisfactory prediction results. Also, the lowest misprediction rate of this benchmark is greater than the highest misprediction rate of jpeg_trace. This shows that the choice of predictor also depends on the incoming instructions and this point must be taken into account which choosing a particular architecture.



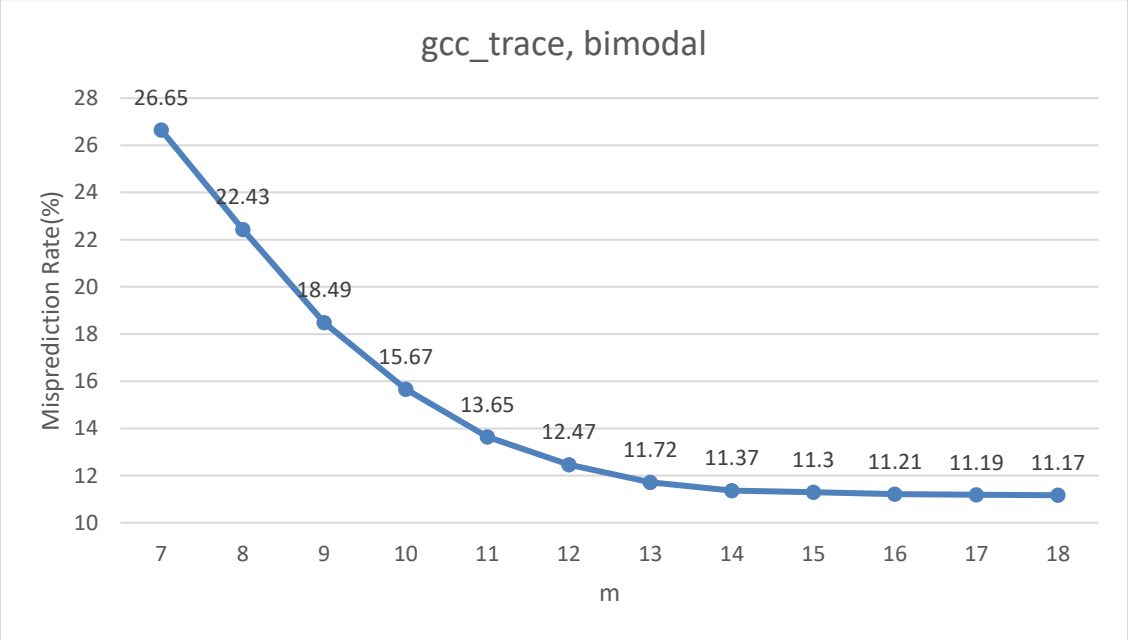
The perl_trace has the highest rate of decrease in misprediction rate till m=10. After that the graph suddenly seems to flatten out for m=11 and then again decrease considerably for m=12. This shows that there are by increasing the index size till m=10, the branches are split up and a single counter corresponds to fewer number of branches. But after going from m=10 to m=11, very few branches are split up. It depends which bit of the branches changes after a common set of bits. The m corresponding to that bit will show a maximum dip in misprediction rate.



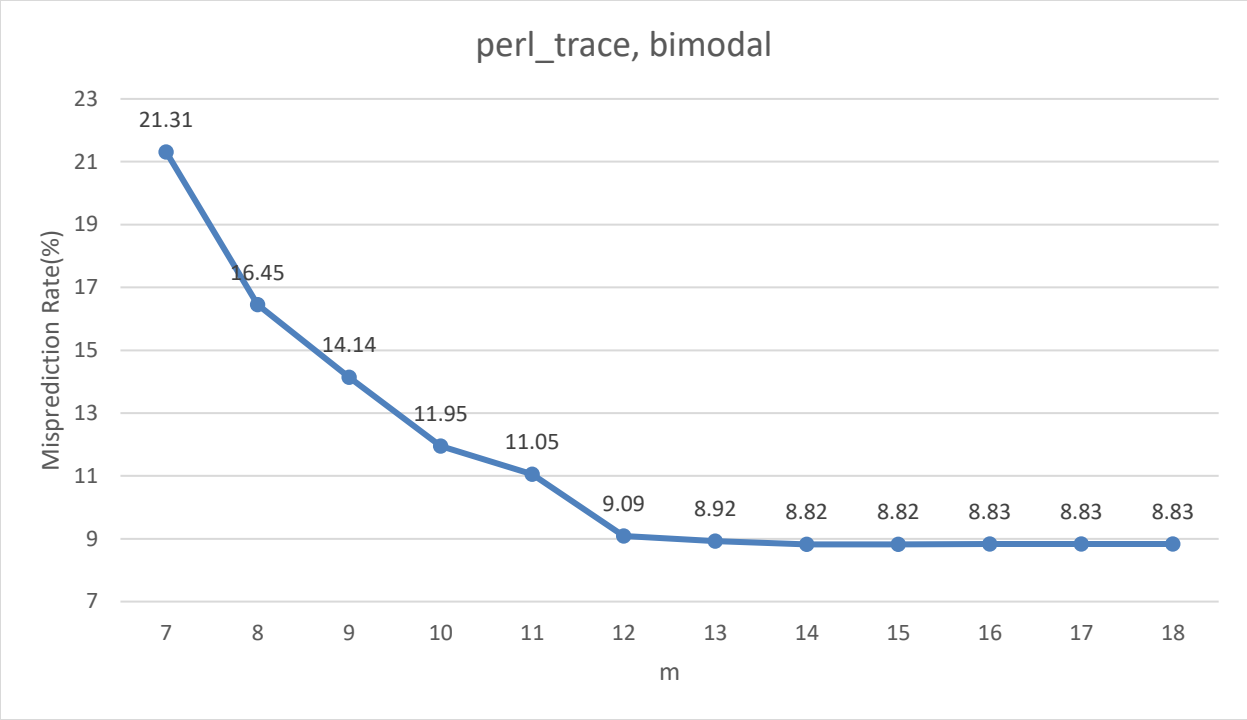
The jpeg_trace has the lowest number of misprediction rate for a particular m among all the traces. This suggests that same branches are repeatedly either being taken or not taken. The curve takes a steeper dip after m=10. This shows that the extra bit in the m=11 simulation separates two frequently occurring branches and thus increases the accuracy of prediction. The same thing is seen happening for the transition from m=7 to m=8.

Finding optimal architecture (minimize misprediction rate and predictor cost):

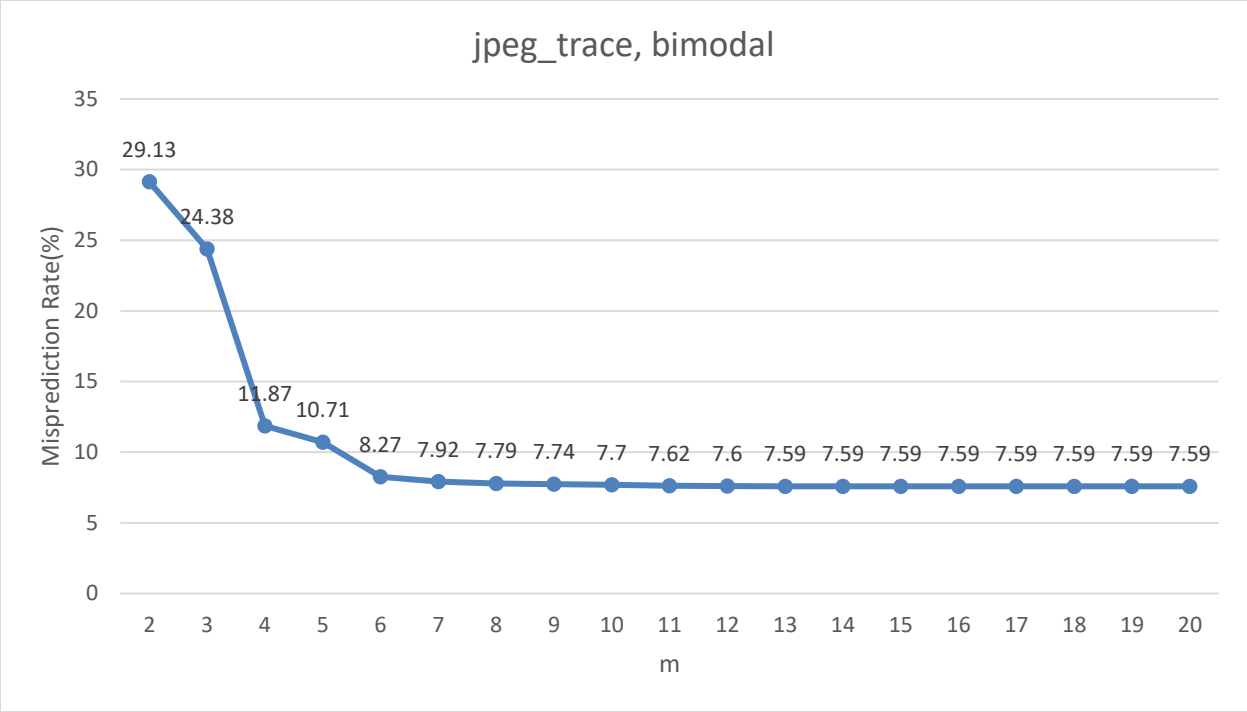
Maximum Budget: 16kB
Therefore, the maximum value of m is
$2*2^m = 16*2^{10}*2^3$
m=16



The graph becomes almost flat at m=13 which has a misprediction rate of 11.72%. Therefore, we can get a minimal misprediction rate, **11.72%** at a reasonable storage of **2kB** with **m=13**.



The slope of the graph becomes almost zero at m=12 where the misprediction rate falls from 11.05 at m=11 to 9.09 at m=12 and then decreases very slowly. Therefore, a good design would be to use a **m=12** architecture having a storage of **500B** and misprediction rate of **9.09%**.



A good design would be where m=6 since the values of misprediction rates become almost constant after this value. Therefore, it would be of no use to increase the hardware cost after this. Hence, the best architecture for this benchmark would be **m=6** with a storage of **16B** with a misprediction rate of **8.27%**.

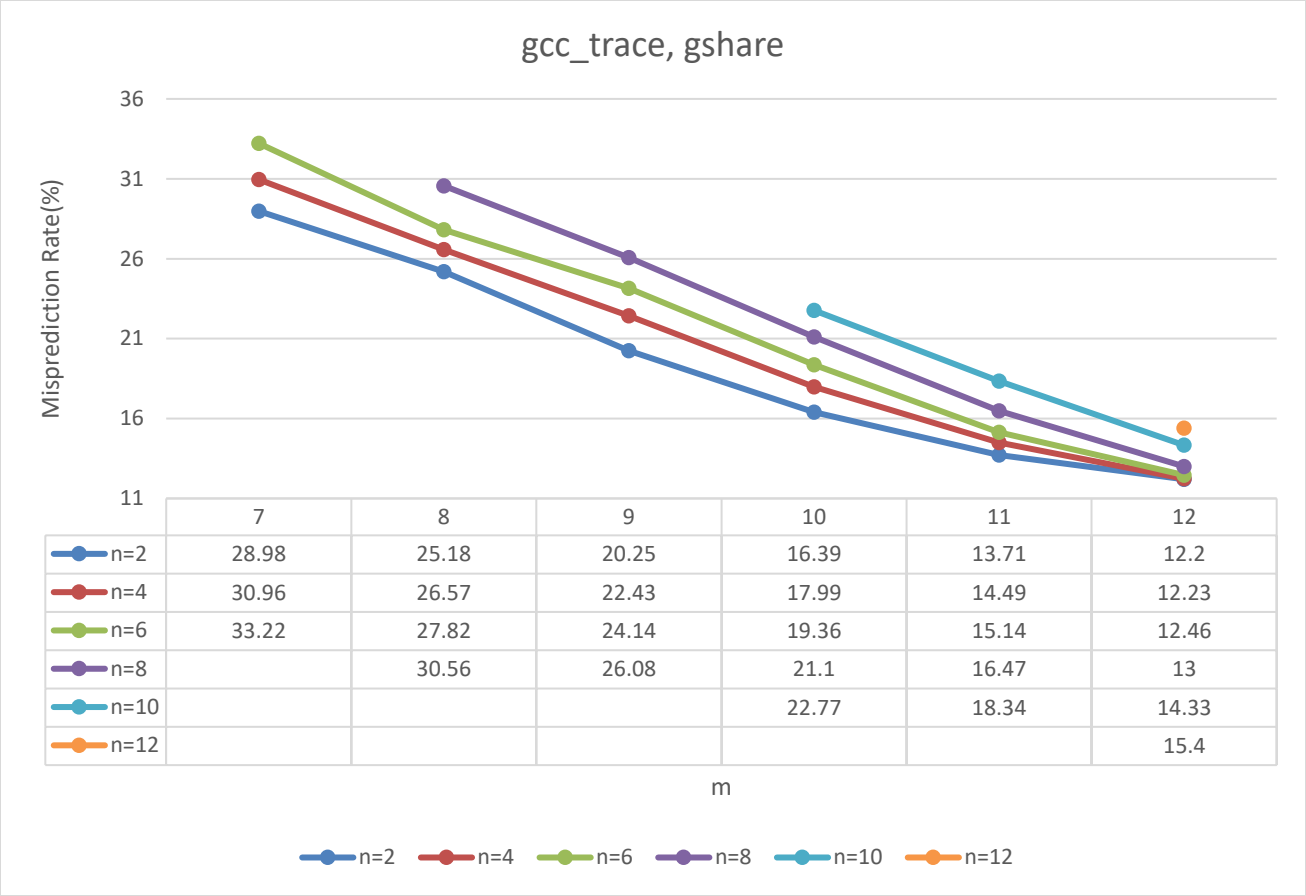
Gshare Predictor:

The difference between a gshare and a bimodal predictor is that a gshare predictor uses a Branch History Register (BHR) which is a multibit shift register. The branch outcome is shifted into this register so that it can be used to better predict the branch outcome if there is a pattern which the branches follow. The bits from the BHR are then hashed with the PC bits to form an index to the prediction table. This PC and BHR combination can also make sure that each combination goes to a different entry in the table. Therefore, the BHR makes sure that if the branches follow a pattern, the pattern is recorded and can be used to predict the upcoming branches unlike bimodal where the only evidence used to predict the branch was its previous few outcomes.

For example, if a single branch keeps flipping between being taken and not being taken, a bimodal predictor will predict based on the last few outcomes of the branch. Whereas a gshare predictor will have the pattern mapped into its BHR which will be hashed with the PC bits to give an index corresponding to the entry of either the branch being taken or not.

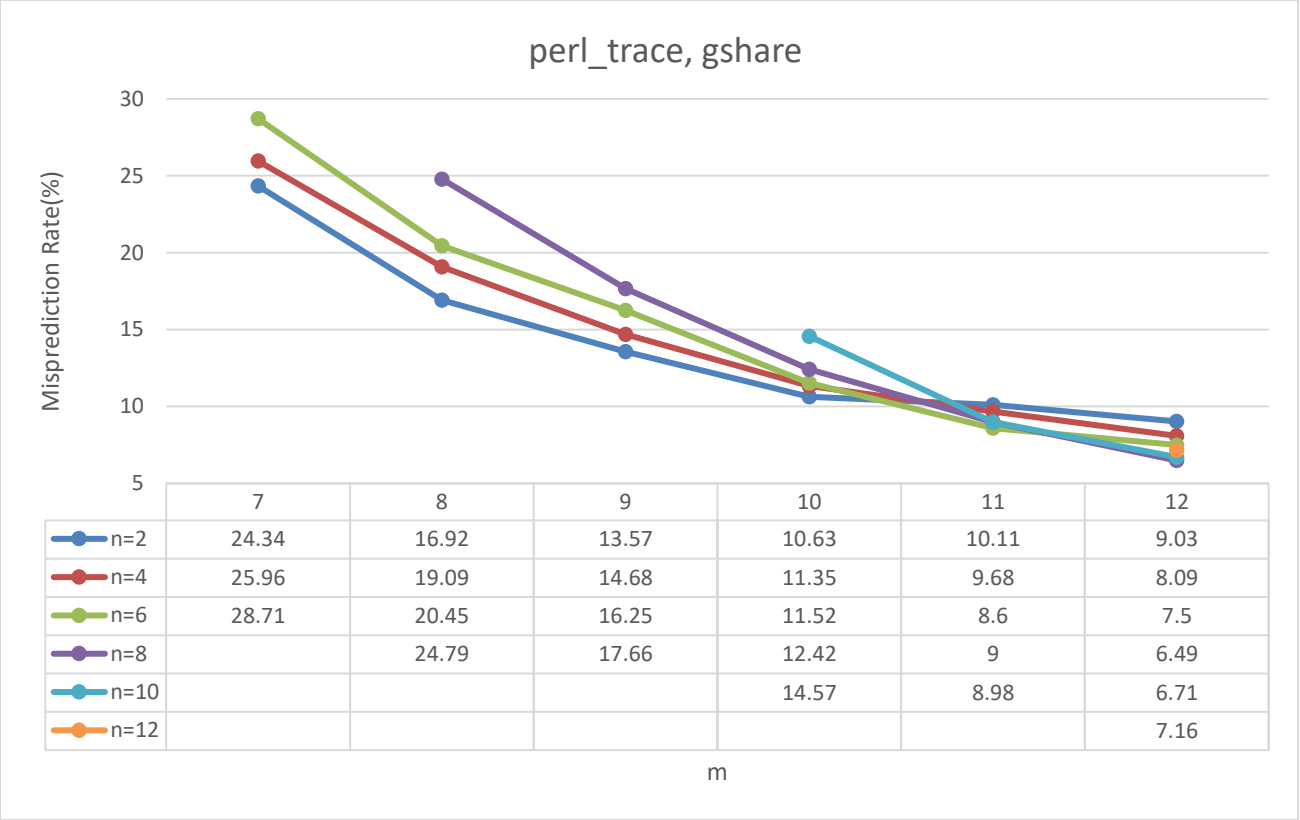
It is better to use bimodal if local history is more beneficial and better to use gshare if global history is more beneficial. A gshare will have more number of mispredictions if many different branches are mixed up.

The graphs plotted from the simulations of the gshare predictor for three different traces are given below.



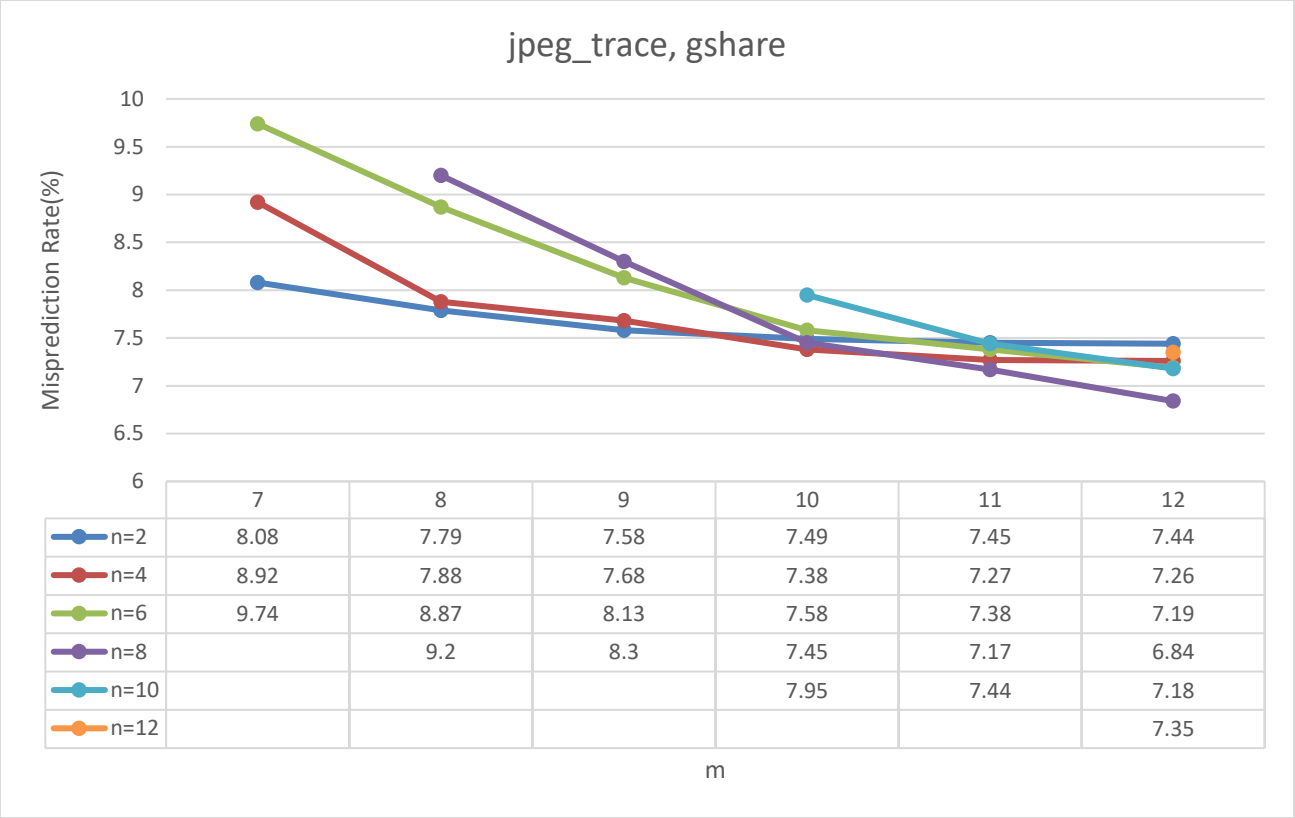
	7	8	9	10	11	12
n=2	28.98	25.18	20.25	16.39	13.71	12.2
n=4	30.96	26.57	22.43	17.99	14.49	12.23
n=6	33.22	27.82	24.14	19.36	15.14	12.46
n=8		30.56	26.08	21.1	16.47	13
n=10				22.77	18.34	14.33
n=12						15.4

As the number of bits (n) used for BHR increase, the misprediction rate goes on increasing for values of m ranging from 7 to 12. It is also seen that misprediction increases at a faster rate for smaller values of m. This shows that many different are mixed up and therefore the BHR stores the history of branches which may not be useful to predict a random different branch. A small value of m also increases interference further increasing the number of mispredictions. This benchmark has the maximum misprediction rate among all the benchmarks. More the number of different branch, more the misprediction rate. Occurrence of same branch many times will decrease the misprediction rate.



	7	8	9	10	11	12
n=2	24.34	16.92	13.57	10.63	10.11	9.03
n=4	25.96	19.09	14.68	11.35	9.68	8.09
n=6	28.71	20.45	16.25	11.52	8.6	7.5
n=8		24.79	17.66	12.42	9	6.49
n=10				14.57	8.98	6.71
n=12						7.16

The misprediction rate in this benchmark keeps increasing on increasing n for values of m from 7 to 10. For m=11, the value first decreases till n=6 and then increases to settle to an almost constant value. For m=12, the value decreases till n=8 and then drastically increases. This benchmark has an irregular variation of misprediction rate with n unlike gcc_trace benchmark. The lowest value of misprediction rate is observed at a configuration of m=12, n=10 for the given calculations. The highest increase in misprediction rate for a particular m is observed at m=8 from n=6 to n=8.

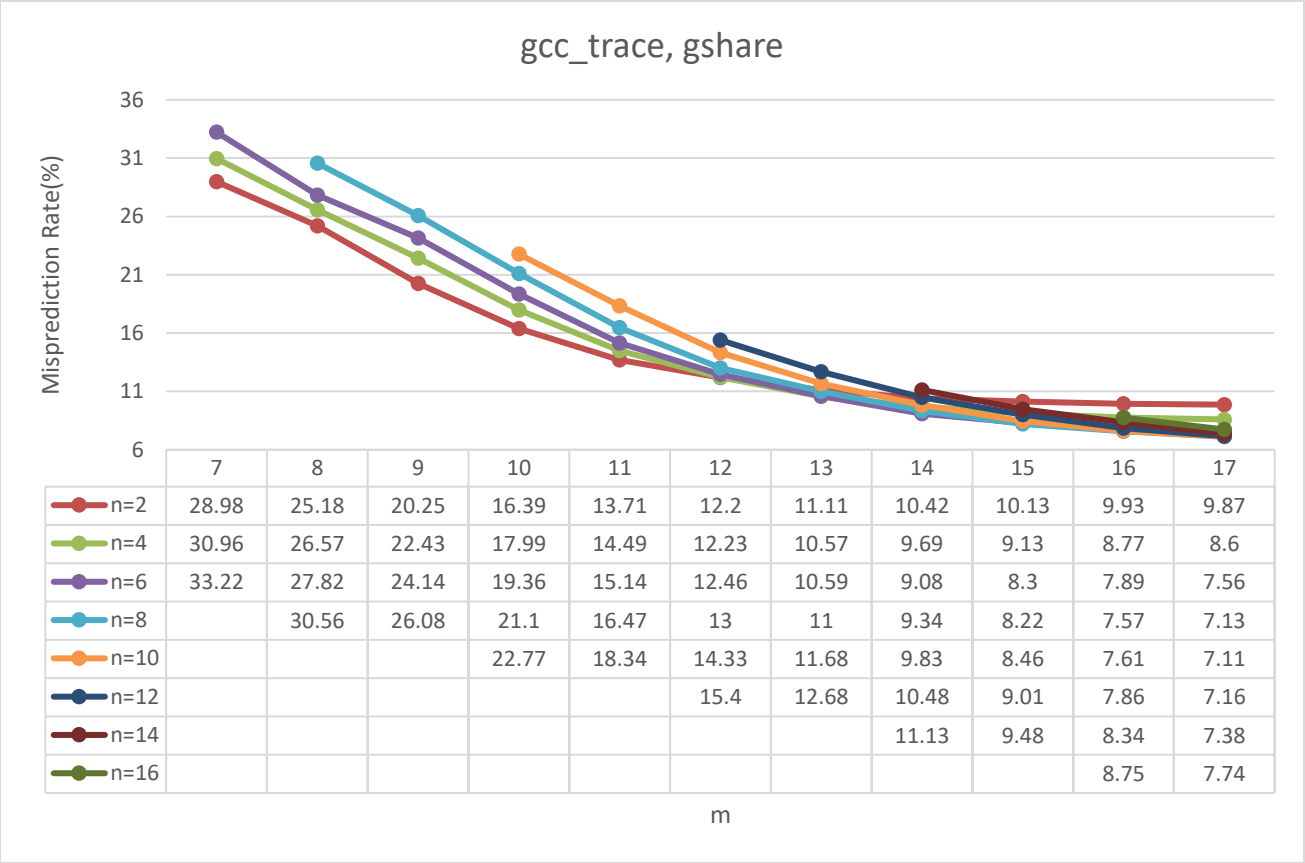


	7	8	9	10	11	12
n=2	8.08	7.79	7.58	7.49	7.45	7.44
n=4	8.92	7.88	7.68	7.38	7.27	7.26
n=6	9.74	8.87	8.13	7.58	7.38	7.19
n=8		9.2	8.3	7.45	7.17	6.84
n=10				7.95	7.44	7.18
n=12						7.35

This benchmark gives the lowest values of misprediction rate among all the benchmarks. The misprediction rate decreases very slowly with increasing value of m. This benchmark shows similar results as compared to the perl benchmark where the misprediction rate increases with increasing n for m=7,8,9. But unlike perl benchmark, the value keeps increasing and decreasing for m=10,11. For m=12, the value decreases till n=8 and then increases again.

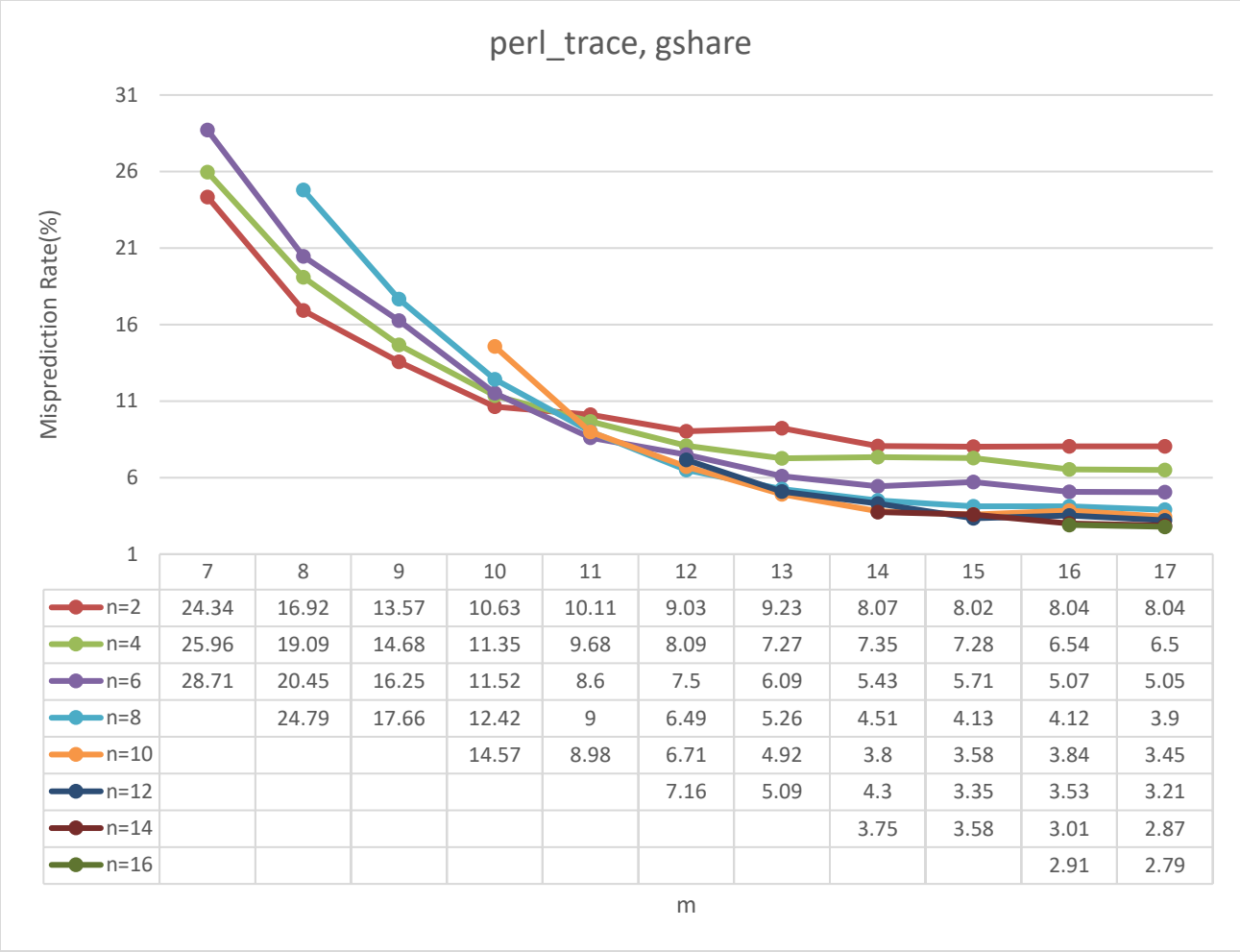
Finding optimal architecture (minimize misprediction rate and predictor cost):

Maximum Budget: 16kB
Therefore, the maximum value of m is
$2*2^m = 16*2^{10}*2^3$
m=16
The number of bits from BHR are ignored since n is a very small number as compared to the storage of the prediction table.



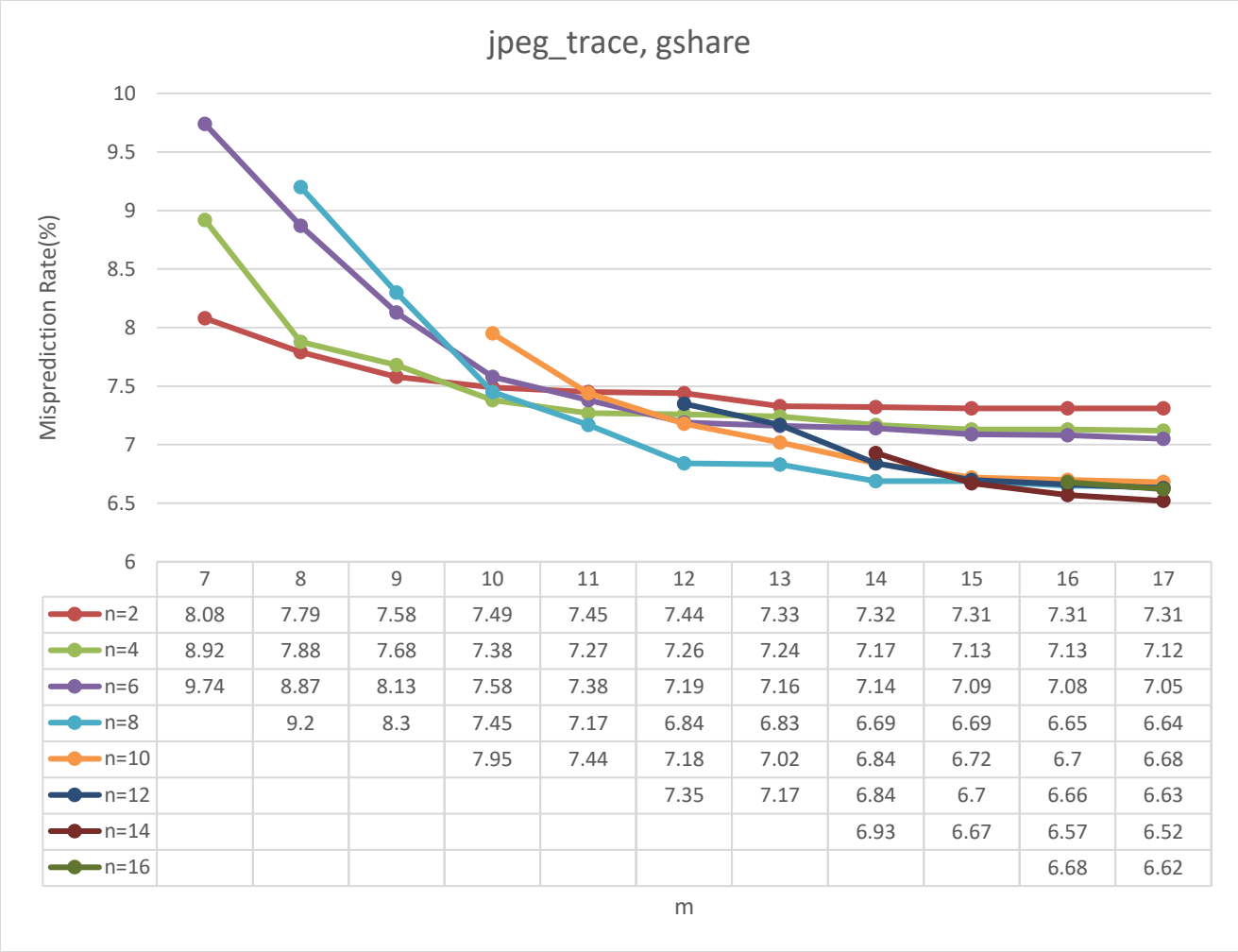
	7	8	9	10	11	12	13	14	15	16	17
n=2	28.98	25.18	20.25	16.39	13.71	12.2	11.11	10.42	10.13	9.93	9.87
n=4	30.96	26.57	22.43	17.99	14.49	12.23	10.57	9.69	9.13	8.77	8.6
n=6	33.22	27.82	24.14	19.36	15.14	12.46	10.59	9.08	8.3	7.89	7.56
n=8		30.56	26.08	21.1	16.47	13	11	9.34	8.22	7.57	7.13
n=10				22.77	18.34	14.33	11.68	9.83	8.46	7.61	7.11
n=12						15.4	12.68	10.48	9.01	7.86	7.16
n=14								11.13	9.48	8.34	7.38
n=16										8.75	7.74

The design m=16, n=8 gives a minimum value of misprediction rate. Although, if we use m=15, n=8 we get a misprediction value which is only a little greater than the one for m=16, n=8. This reduces the predictor cost. Therefore, the optimal design is **m=15, n=8** having a storage of **8kB** and misprediction rate of **8.22%**.



	7	8	9	10	11	12	13	14	15	16	17
n=2	24.34	16.92	13.57	10.63	10.11	9.03	9.23	8.07	8.02	8.04	8.04
n=4	25.96	19.09	14.68	11.35	9.68	8.09	7.27	7.35	7.28	6.54	6.5
n=6	28.71	20.45	16.25	11.52	8.6	7.5	6.09	5.43	5.71	5.07	5.05
n=8		24.79	17.66	12.42	9	6.49	5.26	4.51	4.13	4.12	3.9
n=10				14.57	8.98	6.71	4.92	3.8	3.58	3.84	3.45
n=12						7.16	5.09	4.3	3.35	3.53	3.21
n=14								3.75	3.58	3.01	2.87
n=16										2.91	2.79

A good design for this benchmark would be m=15, n=12. The misprediction rate for m=14, n=12 increases by a little more than 1%. The design m=14, n=10 is also a good candidate and is better than m=15, n=12 because of the reduced predictor cost and only a small difference in misprediction rate. Therefore, the design which minimizes misprediction rate and predictor cost is **m=14, n=10** having a storage of **4kB** and misprediction rate of **3.8%**.



	7	8	9	10	11	12	13	14	15	16	17
n=2	8.08	7.79	7.58	7.49	7.45	7.44	7.33	7.32	7.31	7.31	7.31
n=4	8.92	7.88	7.68	7.38	7.27	7.26	7.24	7.17	7.13	7.13	7.12
n=6	9.74	8.87	8.13	7.58	7.38	7.19	7.16	7.14	7.09	7.08	7.05
n=8		9.2	8.3	7.45	7.17	6.84	6.83	6.69	6.69	6.65	6.64
n=10				7.95	7.44	7.18	7.02	6.84	6.72	6.7	6.68
n=12						7.35	7.17	6.84	6.7	6.66	6.63
n=14								6.93	6.67	6.57	6.52
n=16										6.68	6.62

The graph for n=8 is lowest for mid-range values of m. Therefore, the design with **m=12, n=8** minimizes both the misprediction rate and the predictor cost and has a misprediction rate of **6.84%** with a storage of **1kB**.