**NC State University**

**Department of Electrical and Computer Engineering**

**ECE 463/563: Fall 2021 (Rotenberg)**

# Project #2: Branch Prediction

**by**

**RAMACHANDRAN SEKANIPURAM SRIKANTHAN**

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

Student's electronic signature: Ramachandran Sekanipuram Srikanthan.

Course number: 563

# Part 1: Bimodal Predictor

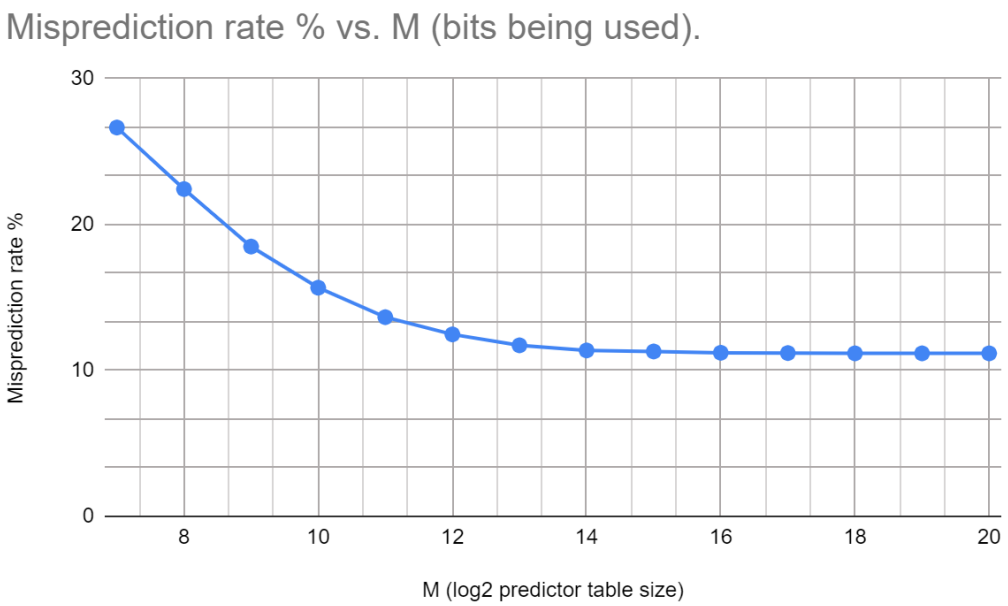
The bimodal predictor maintains a counter table to predict whether the instruction is a branch and not. If it is predicted as branch, then it predicts whether the branch should be taken or not. The predictor uses M bits of the instruction from the PC. Since M bits are being used, the bimodal predictor's counter table has $2^M$ elements in it. Since the lower two bits in RISCV instruction set are 0's it can safely be ignored. Thus, we use the value from the second bit from LSB till M bits from the address traces to find the index of counter table.

The counter table can contain value between 0 and 3. The values 0 and 3 represents strongly not taken and strongly taken respectively. The values 1 and 2 represents weakly not taken and weakly taken respectively. Initially all the elements in counter table are being set to 2 (i.e., weakly taken).

The index calculated from the previous step is used to find the prediction from the counter table. If the index value is greater than or equal to 2, it is predicted as branch taken, else as not taken. The counter table is finally updated based on the actual outcome of the branch. If the actual outcome is taken, the counter table's value is incremented till 3 as maximum. If the actual outcome is not taken, the counter table's value is decremented till 0 as minimum.

## Performance of Bimodal Predictor for "gcc_trace" file

| M | Misprediction rate % |
|---|---|
| 7 | 26.65 |
| 8 | 22.43 |
| 9 | 18.49 |
| 10 | 15.67 |
| 11 | 13.65 |
| 12 | 12.47 |
| 13 | 11.72 |
| 14 | 11.37 |
| 15 | 11.3 |
| 16 | 11.21 |
| 17 | 11.19 |
| 18 | 11.17 |
| 19 | 11.17 |
| 20 | 11.17 |



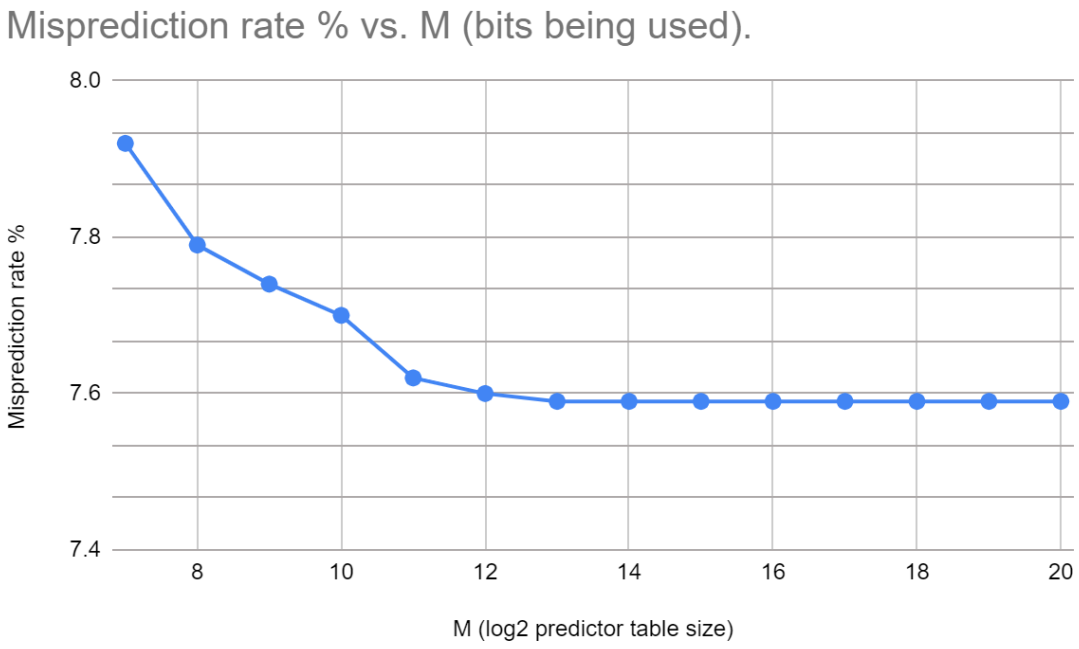

From the graph we can find out that, the misprediction rate decreases as we increase the value of M which is directly affects the predictor table size. As M increases the number of entries in the predictor table increases and so we have a lot of space to store more information. It can also be seen that from the gcc_trace.txt file the number of addresses targeting the same index in the prediction table is higher for small values of M, thus increasing the misprediction rate. Thus, when M is increased, we have a few addresses targeting the same index in the prediction table thereby decreasing the misprediction rate.

After a particular value of M (M = 12 in this case), the misprediction rate saturates and the rates don't diverge much as M increases. The reason for this behavior is that after a certain value of M, most of the addresses map to a unique location and so there is less collision between various addresses, thus making misprediction rate depend purely on the previously trained data causing saturation of misprediction rate.

Thus, for M = 12, the trade off between predictor table size and misprediction rate provides the best value.

## Performance of Bimodal Predictor for “jpeg_trace” file

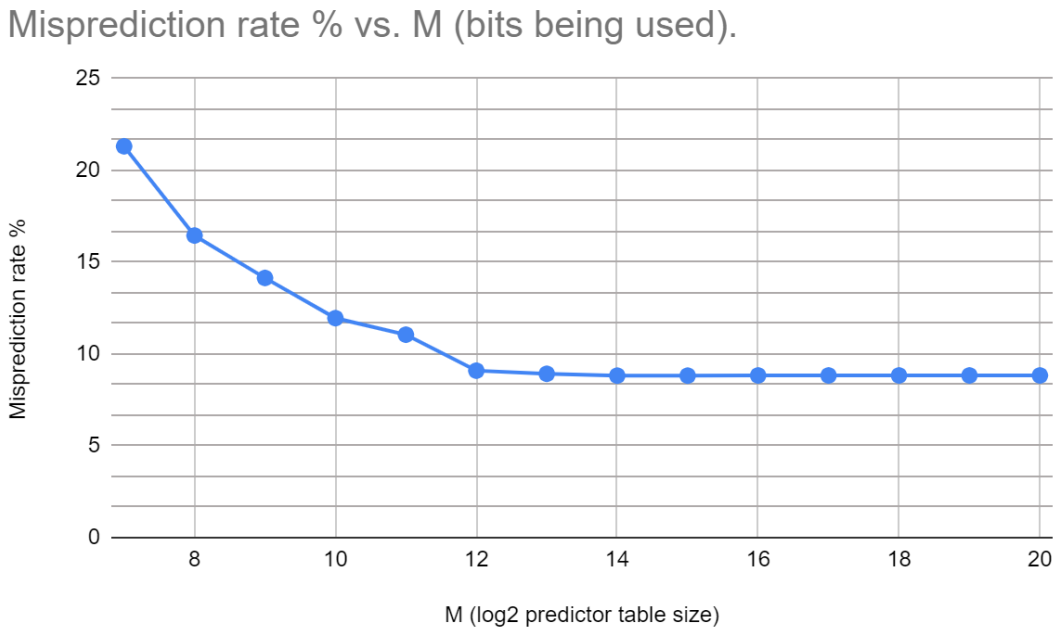
| M | Misprediction rate % |
|---|---|
| 7 | 7.92 |
| 8 | 7.79 |
| 9 | 7.74 |
| 10 | 7.7 |
| 11 | 7.62 |
| 12 | 7.6 |
| 13 | 7.59 |
| 14 | 7.59 |
| 15 | 7.59 |
| 16 | 7.59 |
| 17 | 7.59 |
| 18 | 7.59 |
| 19 | 7.59 |
| 20 | 7.59 |



From the graph, we can see that the misprediction rate decreases as we increase the predictor table size (increasing M bits) and finally saturates after a certain value of M. We can also note from the graph that, the misprediction rate decreases steadily until it reaches a saturation value, indicating that most of addresses map to a unique index in the predictor table thereby making misprediction rate depend mainly on the previously trained data. The point of M = 11, gives the best value for tradeoff between the prediction table size in memory and the misprediction rate.

## Performance of Bimodal Predictor for "perl_trace" file

| M | Misprediction rate % |
|---|---|
| 7 | 21.31 |
| 8 | 16.45 |
| 9 | 14.14 |
| 10 | 11.95 |
| 11 | 11.05 |
| 12 | 9.09 |
| 13 | 8.92 |
| 14 | 8.82 |
| 15 | 8.82 |
| 16 | 8.83 |
| 17 | 8.83 |
| 18 | 8.83 |
| 19 | 8.83 |
| 20 | 8.83 |




The graph indicates that as we increase the value of M the misprediction rate decreases until a certain M value and becomes saturated after that. It can be seen clearly from the “perl_trace” dataset that the misprediction rate is very much high for lower values of M and a slightly increases causes a drastic increase in performance. This means that there are lot of address traces mapping to the same target index when very few bits are chosen to find out the target index. With a slight increase in M value, much of the address traces maps to a unique target index. The value of M = 12 gives a best value for tradeoff between memory occupancy for storing the predictor table and the misprediction rate.

## Comparison among various benchmarks

From all the above benchmarks we can clearly see that the misprediction rate decreases as we increase the size of M causing a large size of predictor table in memory until a point and after that the value saturates. The point before which the misprediction rate saturates is the best point for handling the tradeoff between memory size needed to store the prediction table and misprediction rate.

We can also note that, the “gcc_trace” and “perl_trace” has a misprediction rate significantly higher than the “jpeg_trace” benchmark indicating that, the “jpeg_trace” benchmark has a lot of unique address trace, thus enabling the misprediction rate to depend mainly on previously trained data.

From the graphs, we can also note that for “gcc_trace” and “perl_trace” benchmarks the curve saturates for M = 12 and for “jpeg_trace” the curve saturates at M = 11 indicating the uniqueness in the address traces in the “jpeg_trace” benchmark.

## Part 2: G-Share Predictor

The G-Share predictor also utilizes a predictor table to store and train from the previously resulted outcomes. The predictor uses M bits from the address trace and creates a table of size $2^M$ to store the predictions. The lower two bits of the address traces are ignored as they are 0's for RISCV instruction set. The Predictor also utilizes a global branch history register to keep in track of branch operation in a global scope. The global history register stores the value of the actual outcomes of the branches and can remember up to N bits.

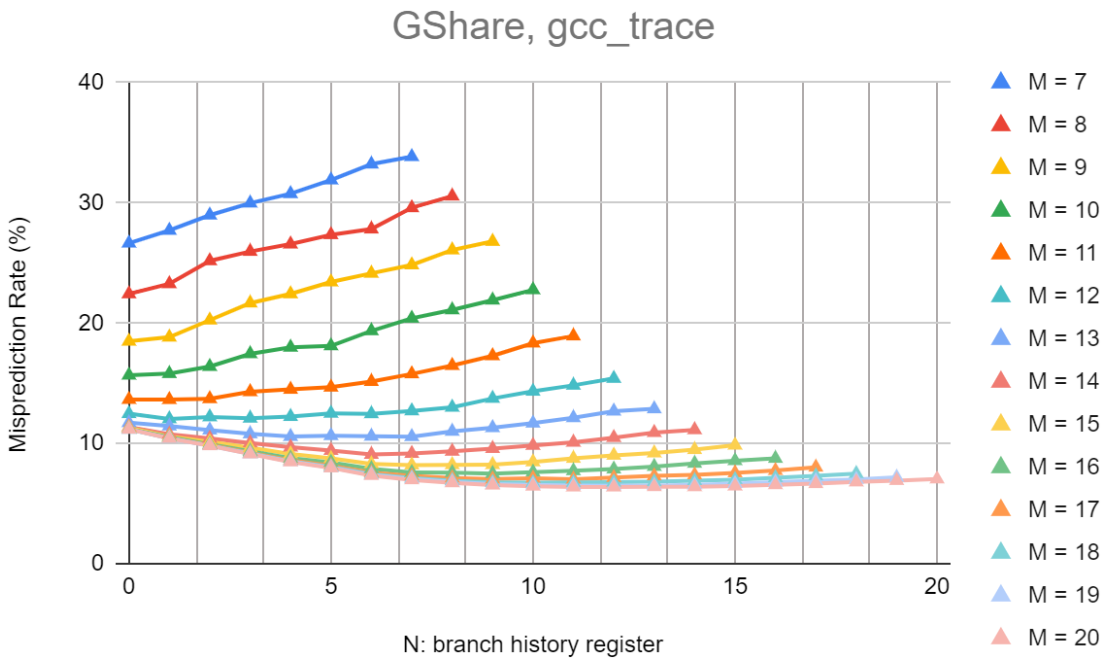
The Predictor table can be in any of the four values between 0 and 3 at any time. The values 0 and 3 represents strongly not taken and strongly taken respectively. The values 1 and 2 represents weakly not taken and weakly taken respectively. The predictor table is initialized to 2 (weakly taken). When the outcome of the corresponding address trace indicated by the index of the predictor table is taken, then the value in the table is incremented by 1 up to a maximum value of 3. When the outcome of the corresponding address trace indicated by the index of the predictor table is not taken, then the value in the table is decremented by 1 up to a minimum value of 0.

The index is calculated by using a hashing function which maps the address traces to a corresponding range of hash values which can be used to index into the predictor table. The last N bits (from MSB) of the address from the trace is XORed with the N bits of global branch history register's contents. This result is then concatenated with the remaining M-N bits from the address trace to map into the index of the predictor table.

Depending on the actual outcome of the branch the global history register is also updated. The register is right shifted by 1 and the value of 1 is appended into the MSB position if the actual outcome is a branch taken, else a value of 0 is appended into the MSB.

## Performance of G-Share Predictor for “gcc_trace” file

| N | M = 7 | M = 8 | M = 9 | M = 10 | M = 11 | M = 12 | M = 13 | M = 14 | M = 15 | M = 16 | M = 17 | M = 18 | M = 19 | M = 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 26.65 | 22.43 | 18.49 | 15.67 | 13.65 | 12.47 | 11.72 | 11.37 | 11.3 | 11.21 | 11.19 | 11.17 | 11.17 | 11.17 |
| **1** | 27.71 | 23.27 | 18.83 | 15.8 | 13.64 | 12.04 | 11.44 | 10.76 | 10.58 | 10.52 | 10.44 | 10.43 | 10.42 | 10.42 |
| **2** | 28.98 | 25.18 | 20.25 | 16.39 | 13.71 | 12.2 | 11.11 | 10.42 | 10.13 | 9.93 | 9.87 | 9.82 | 9.81 | 9.8 |
| **3** | 29.98 | 25.96 | 21.67 | 17.45 | 14.29 | 12.09 | 10.79 | 10.04 | 9.64 | 9.34 | 9.22 | 9.18 | 9.13 | 9.12 |
| **4** | 30.76 | 26.57 | 22.43 | 17.99 | 14.49 | 12.23 | 10.57 | 9.69 | 9.13 | 8.77 | 8.6 | 8.52 | 8.5 | 8.46 |
| **5** | 31.9 | 27.35 | 23.42 | 18.11 | 14.67 | 12.5 | 10.64 | 9.41 | 8.75 | 8.42 | 8.16 | 8.06 | 7.99 | 7.96 |
| **6** | 33.22 | 27.82 | 24.14 | 19.36 | 15.14 | 12.46 | 10.59 | 9.08 | 8.3 | 7.89 | 7.56 | 7.45 | 7.38 | 7.33 |
| **7** | 33.84 | 29.59 | 24.84 | 20.4 | 15.78 | 12.7 | 10.56 | 9.16 | 8.2 | 7.58 | 7.3 | 7.12 | 7.05 | 6.97 |
| **8** | | 30.56 | 26.08 | 21.1 | 16.47 | 13 | 11 | 9.34 | 8.22 | 7.57 | 7.13 | 6.91 | 6.79 | 6.72 |
| **9** | | | 26.79 | 21.91 | 17.28 | 13.74 | 11.3 | 9.57 | 8.23 | 7.49 | 7.04 | 6.76 | 6.61 | 6.53 |
| **10** | | | | 22.77 | 18.34 | 14.33 | 11.68 | 9.83 | 8.46 | 7.61 | 7.11 | 6.73 | 6.55 | 6.44 |
| **11** | | | | | 18.93 | 14.83 | 12.13 | 10.09 | 8.75 | 7.72 | 7.03 | 6.75 | 6.5 | 6.37 |
| **12** | | | | | | 15.4 | 12.68 | 10.48 | 9.01 | 7.86 | 7.16 | 6.78 | 6.47 | 6.37 |
| **13** | | | | | | | 12.87 | 10.9 | 9.2 | 8.08 | 7.3 | 6.81 | 6.5 | 6.4 |
| **14** | | | | | | | | 11.13 | 9.48 | 8.34 | 7.38 | 6.9 | 6.55 | 6.39 |
| **15** | | | | | | | | | 9.85 | 8.56 | 7.55 | 6.99 | 6.66 | 6.45 |
| **16** | | | | | | | | | | 8.75 | 7.74 | 7.15 | 6.78 | 6.56 |
| **17** | | | | | | | | | | | 8.02 | 7.3 | 6.9 | 6.65 |
| **18** | | | | | | | | | | | | 7.49 | 7.01 | 6.79 |
| **19** | | | | | | | | | | | | | 7.18 | 6.91 |
| **20** | | | | | | | | | | | | | | 7.04 |




From the graph we can understand that, the size of global history registers also has an impact in the misprediction rate. For larger values of M, as the size of the global history register is increased there is a significant drop in the misprediction rate until a certain value. Now for a fixed value of M, we can see that as we increase the size of global history register (N), there is a significant decrease in the misprediction rate. The reason being that smaller values of N can store less global branch taken outcome. The XOR in the hashing function ensures that this previous global branch taken data is taken into consideration while calculating the index into the predictor table. The concatenation operation with the remaining M-N bits of the current trace is also used in the calculation of the index into the table.

Now if we increase the value of N, the misprediction rate decreases as we have more global data taken into consideration. This also means that we have sufficient M-N bits concatenated, thereby providing sufficient current address trace's data. But if the N increases beyond a value, we start to accumulate large amount of older data which may not be of significant help in the current prediction. Due to this accumulation of older data, the current prediction gets biased to all the older values causing the misprediction rate to increase. Also, M-N becomes smaller as we increase N causing a much biasing to previous outcomes by giving less weight to present address trace's contents.

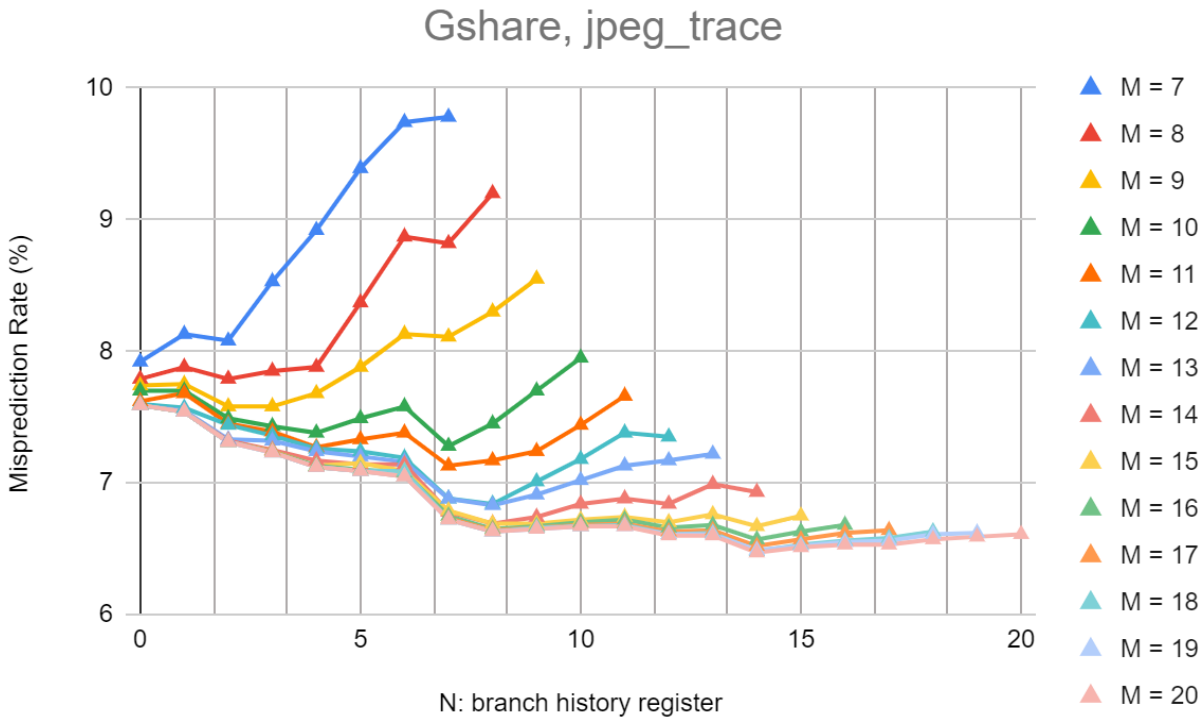
We can also see that, for smaller values of M, increasing N has a negative impact in the performance. Since for smaller values of M we may have less indexes in the table and so there may be much of collisions into the same index. In that case when we increase the N, much of older values of branch history gets accumulated giving less space (M-N bits) to take into consideration of the present address's traces which of significantly smaller in size thereby increasing the misprediction rate.

Thus, for G-Share predictor we need a storage overhead is $N + 2^M$ for both predictor table and global history register. The values of M and N for which we have an optimal predictor table size and global history register size for which the misprediction

rate is not too much higher is for M=14 and N=6. Since above which for the same value of N we can see very slight decrease in the misprediction rate for which the storage overhead becomes exponentially high in powers of 2. Also, for large values of N we can see the misprediction rate goes higher as discussed before.

## Performance of G-Share Predictor for "jpeg_trace" file

| N | M = 7 | M = 8 | M = 9 | M = 10 | M = 11 | M = 12 | M = 13 | M = 14 | M = 15 | M = 16 | M = 17 | M = 18 | M = 19 | M = 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.92 | 7.79 | 7.74 | 7.7 | 7.62 | 7.6 | 7.59 | 7.59 | 7.59 | 7.59 | 7.59 | 7.59 | 7.59 | 7.59 |
| **1** | 8.13 | 7.88 | 7.75 | 7.7 | 7.68 | 7.57 | 7.55 | 7.54 | 7.54 | 7.54 | 7.54 | 7.54 | 7.54 | 7.54 |
| **2** | 8.08 | 7.79 | 7.58 | 7.49 | 7.45 | 7.44 | 7.33 | 7.32 | 7.31 | 7.31 | 7.31 | 7.31 | 7.31 | 7.31 |
| **3** | 8.53 | 7.85 | 7.58 | 7.43 | 7.39 | 7.36 | 7.32 | 7.25 | 7.24 | 7.24 | 7.24 | 7.23 | 7.23 | 7.23 |
| **4** | 8.92 | 7.88 | 7.68 | 7.38 | 7.27 | 7.26 | 7.24 | 7.17 | 7.13 | 7.13 | 7.12 | 7.12 | 7.12 | 7.12 |
| **5** | 9.39 | 8.37 | 7.88 | 7.49 | 7.33 | 7.24 | 7.2 | 7.14 | 7.15 | 7.1 | 7.09 | 7.09 | 7.09 | 7.09 |
| **6** | 9.74 | 8.87 | 8.13 | 7.58 | 7.38 | 7.19 | 7.16 | 7.14 | 7.09 | 7.08 | 7.05 | 7.09 | 7.05 | 7.05 |
| **7** | 9.78 | 8.82 | 8.11 | 7.28 | 7.13 | 6.88 | 6.88 | 6.78 | 6.79 | 6.75 | 6.73 | 6.72 | 6.72 | 6.72 |
| **8** | | 9.2 | 8.3 | 7.45 | 7.17 | 6.84 | 6.83 | 6.69 | 6.69 | 6.65 | 6.64 | 6.63 | 6.63 | 6.63 |
| **9** | | | 8.55 | 7.7 | 7.24 | 7.01 | 6.91 | 6.74 | 6.69 | 6.67 | 6.66 | 6.66 | 6.65 | 6.65 |
| **10** | | | | 7.95 | 7.44 | 7.18 | 7.02 | 6.84 | 6.72 | 6.7 | 6.68 | 6.67 | 6.67 | 6.67 |
| **11** | | | | | 7.66 | 7.38 | 7.13 | 6.88 | 6.74 | 6.72 | 6.69 | 6.68 | 6.67 | 6.67 |
| **12** | | | | | | 7.35 | 7.17 | 6.84 | 6.7 | 6.66 | 6.63 | 6.61 | 6.61 | 6.6 |
| **13** | | | | | | | 7.22 | 6.99 | 6.76 | 6.68 | 6.64 | 6.62 | 6.61 | 6.6 |
| **14** | | | | | | | | 6.93 | 6.67 | 6.57 | 6.52 | 6.48 | 6.49 | 6.47 |
| **15** | | | | | | | | | 6.75 | 6.63 | 6.57 | 6.53 | 6.52 | 6.51 |
| **16** | | | | | | | | | | 6.68 | 6.62 | 6.56 | 6.55 | 6.53 |
| **17** | | | | | | | | | | | 6.64 | 6.58 | 6.56 | 6.53 |
| **18** | | | | | | | | | | | | 6.63 | 6.61 | 6.57 |
| **19** | | | | | | | | | | | | | 6.62 | 6.59 |
| **20** | | | | | | | | | | | | | | 6.61 |




The graph indicates that misprediction rate decreases for increasing N in large values of M. The misprediction rate goes much lower for larger M values but at the cost of very large memory overhead for storing predictor table. The M and N values for which we get better performance in terms of misprediction rate with having the lesser memory overhead is for M = 12 and N = 7.

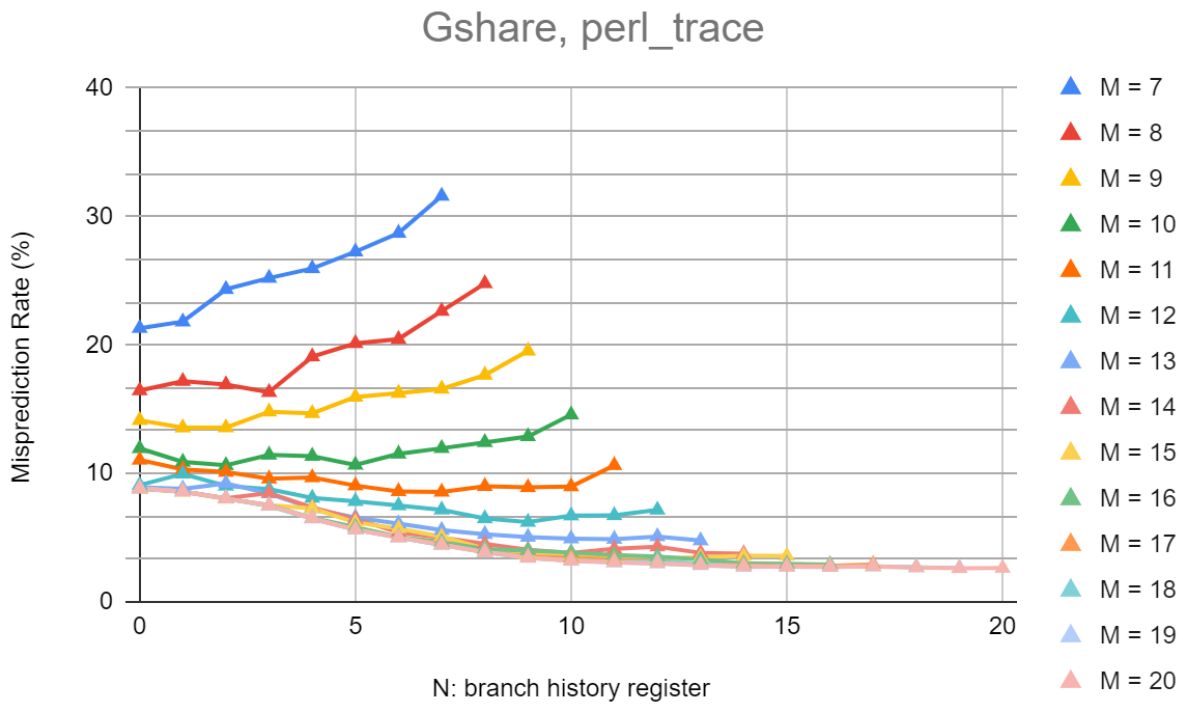
For a particular value of M (Large M value) as N increases, we can have large amount of previous global branch data up to a point and after that with increase in N the accumulation of global history data becomes higher and reduces the usage of current address trace's data thereby increasing misprediction rate. For lower values of M, increasing N causes still lowering of current address's traces values which is already small. Also, the size of predictor table is very small causing lot of mispredictions.

Here we can also see that the misprediction rate for bimodal predictor (n = 0) is lesser when compared to g-share as we increase N values for smaller m values. Thus, it is better to use bimodal predictor for small M values and G-share for large M values.

Now for large values of M, for a particular N, as we increase the value of M, we can only see a slight decrease in misprediction rate at the cost of large increase in memory overhead. Even if we increase N for large values of M, we can see curve nearly overlaps and there is very less change in the misprediction rate for M values from 12 to 20, so its better to have M = 12 as best value without compromising the tradeoff.

## Performance of G-Share Predictor for “perl_trace” file

| N | M = 7 | M = 8 | M = 9 | M = 10 | M = 11 | M = 12 | M = 13 | M = 14 | M = 15 | M = 16 | M = 17 | M = 18 | M = 19 | M = 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 21.31 | 16.45 | 14.14 | 11.95 | 11.05 | 9.09 | 8.92 | 8.82 | 8.82 | 8.83 | 8.83 | 8.83 | 8.83 | 8.83 |
| **1** | 21.81 | 17.18 | 13.56 | 10.89 | 10.32 | 9.97 | 8.8 | 8.6 | 8.56 | 8.6 | 8.59 | 8.59 | 8.59 | 8.59 |
| **2** | 24.34 | 16.92 | 13.57 | 10.63 | 10.11 | 9.03 | 9.23 | 8.07 | 8.02 | 8.04 | 8.04 | 8.04 | 8.04 | 8.04 |
| **3** | 25.21 | 16.32 | 14.8 | 11.44 | 9.58 | 8.77 | 8.44 | 8.48 | 7.52 | 7.52 | 7.49 | 7.46 | 7.49 | 7.49 |
| **4** | 25.96 | 19.09 | 14.68 | 11.35 | 9.68 | 8.09 | 7.27 | 7.35 | 7.28 | 6.54 | 6.5 | 6.48 | 6.5 | 6.5 |
| **5** | 27.26 | 20.13 | 15.95 | 10.66 | 9.05 | 7.83 | 6.57 | 6.44 | 6.18 | 5.82 | 5.62 | 5.62 | 5.62 | 5.62 |
| **6** | 28.71 | 20.45 | 16.25 | 11.52 | 8.6 | 7.5 | 6.09 | 5.43 | 5.71 | 5.07 | 5.05 | 5.02 | 5.01 | 5.01 |
| **7** | 31.6 | 22.64 | 16.58 | 11.96 | 8.56 | 7.17 | 5.58 | 5 | 5.08 | 4.66 | 4.48 | 4.44 | 4.44 | 4.44 |
| **8** | | 24.79 | 17.66 | 12.42 | 9 | 6.49 | 5.26 | 4.51 | 4.13 | 4.12 | 3.9 | 3.86 | 3.89 | 3.86 |
| **9** | | | 19.54 | 12.88 | 8.91 | 6.21 | 5.05 | 4.05 | 3.72 | 3.99 | 3.52 | 3.42 | 3.44 | 3.42 |
| **10** | | | | 14.57 | 8.98 | 6.71 | 4.92 | 3.8 | 3.58 | 3.84 | 3.45 | 3.22 | 3.21 | 3.2 |
| **11** | | | | | 10.63 | 6.73 | 4.89 | 4.13 | 3.42 | 3.64 | 3.28 | 3.13 | 3.06 | 3.08 |
| **12** | | | | | | 7.16 | 5.09 | 4.3 | 3.35 | 3.53 | 3.21 | 3.17 | 2.98 | 3.01 |
| **13** | | | | | | | 4.77 | 3.82 | 3.48 | 3.3 | 2.93 | 2.95 | 2.84 | 2.87 |
| **14** | | | | | | | | 3.75 | 3.58 | 3.01 | 2.87 | 2.78 | 2.71 | 2.75 |
| **15** | | | | | | | | | 3.59 | 2.97 | 2.81 | 2.77 | 2.73 | 2.73 |
| **16** | | | | | | | | | | 2.91 | 2.79 | 2.73 | 2.72 | 2.71 |
| **17** | | | | | | | | | | | 2.93 | 2.76 | 2.76 | 2.76 |
| **18** | | | | | | | | | | | | 2.72 | 2.68 | 2.67 |
| **19** | | | | | | | | | | | | | 2.66 | 2.61 |
| **20** | | | | | | | | | | | | | | 2.63 |




From the graph, we can see that for large values of M as we increase the value of N, much of global branch history is stored and is being used to predict the next outcome which thereby decreases the misprediction rate.

The best value of M and N for having an appreciable predictor table size (appreciable memory overhead) without affect too much of misprediction rate is for M = 14 and N = 10. We can further increase the value of M but the misprediction rate changes slightly but the cost of memory overhead becomes very much higher.

For a fixed value of M as we increase the N size, the history values of branches are given much weightage than the present address's traces causing increase in misprediction rate. Also, for lower values of M, the size of prediction table is small causing increase in misprediction rate.

Also, as we increase N above a point, for a particular M value, the misprediction rate increases causing degradation in predictor performance.

## Comparison among various benchmarks

We can identify that for "gcc_trace" and "perl_trace" we have a value of M = 14 without biasing towards any side in the tradeoff between memory overhead and misprediction rate. For "jpeg_trace" we have a value of M = 12 indicating that this benchmark has lot of address traces mapping to unique target locations in the predictor table.

We can also note that for "jpeg_trace", the bimodal predictor is not able to acquire the misprediction rate of below 7 even for M = 20, but G-share acquires misprediction rate of below 7 for M = 12 keeping N = 7. This shows that N (global history register size) has an impact in misprediction ratio. In the case of bimodal, as the

curve saturates the misprediction rate mainly depends on previously trained data for the same addresses, but here in G-share we can see that, it depends on global branch history along with previously trained data causing the misprediction rate to further reduce which cannot be obtained in case of bimodal predictor.

But for the same “jpeg_trace” benchmark, for lower values of M, bimodal predictor gives the least misprediction rate which cannot be obtained by G-Share predictor for any N as consideration of the branch taken history further negatively impacts the prediction for small M values.

Thus, having a bimodal predictor for lower M values and having a G-share predictor for higher M values with sufficient N provides better results without taking too much of memory overhead and without increasing the misprediction rate.