Project 2: GShare Branch Predictor

by Tyler Townsend[1]
Univeristy of Central Florida[1]

November 28th, 2017

Contents

List of Figures

List of Tables

1 Introduction

As pipelines become deeper, branch delays become costly for CPI. To reduce the CPI and improve the performance of the CPU, branch prediction may be invoked to overcome costly branch delays. Two types of predictions schemes are used: static schemes which rely on information available during compilation, and dynamic scheme which relies on program behavior at runtime [1]. Dynamic Branch Prediction relies on a prediction table, or a Pattern History Table (PHT) which transcribes the behavior of previous branches. The PHT may use a 1 or 2-bit history scheme and each index is addressed with a PC index and a history register (HR) index. The history register index records the most recent branches.

A popular dynamic predictor design is GShare. The PHT uses a HR that shares the global history between all addresses and is used for adaptive prediction. The implementation for the GShare prediction can be seen in Figure 1.

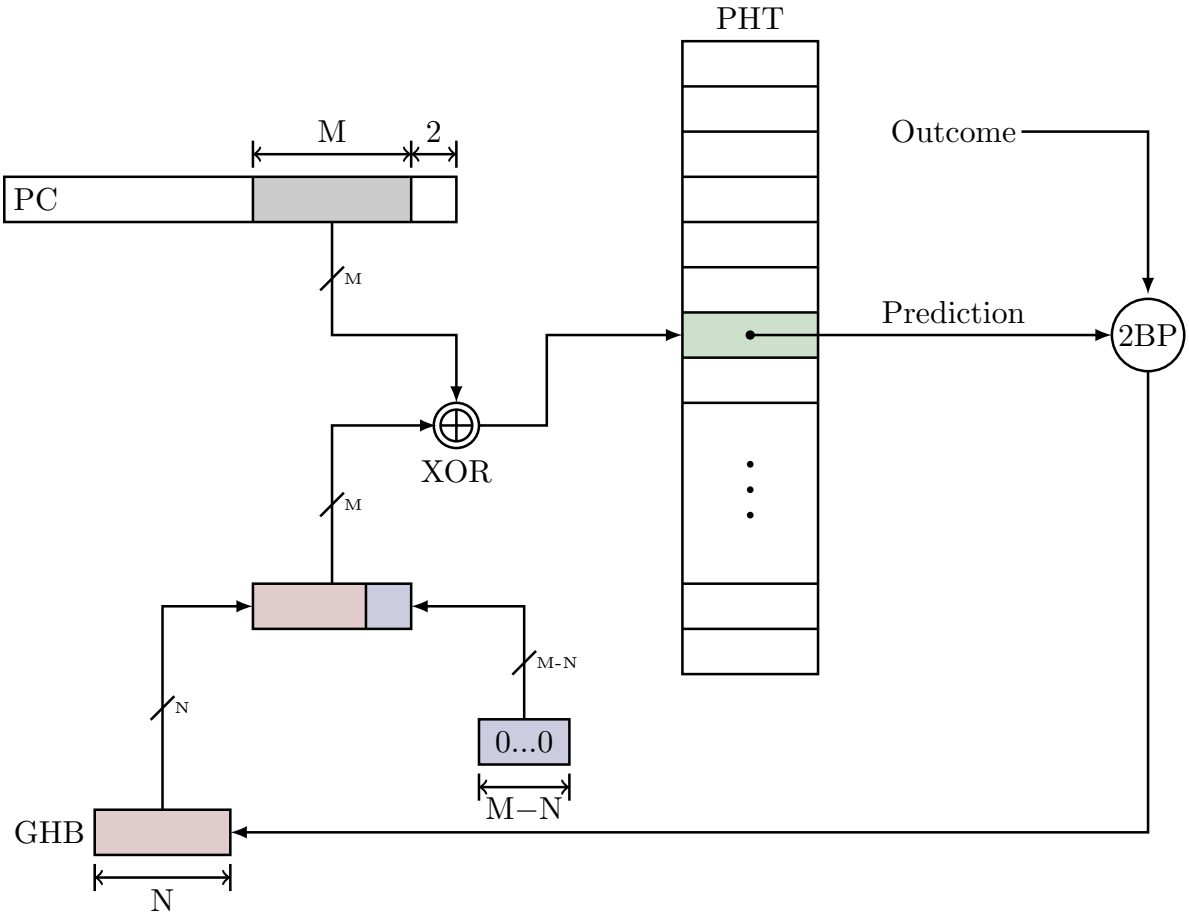


Figure 1: GShare Branch Predictor Diagram used for this study

The program counter (PC) instruction is read in and the first two bits are removed from the address. Then M bits are taken from the address. These M bits correspond to the size of the PHT which

is 2^M. The global history buffer (GHB), the history register, stores the previous N branches' history. $M - N$ 0's are padded onto the GHB and the results is XORed with M bits from the PC address. This is used to lookup the branch prediction in the PHT. This prediction is compared to the outcome using the 2-bit predictor (2BP) scheme as seen in Figure 2. The prediction is updated and the history is stored in the GHB by removing the LSB and adding a 1 (if taken) or 0 (if not taken) as the MSB.

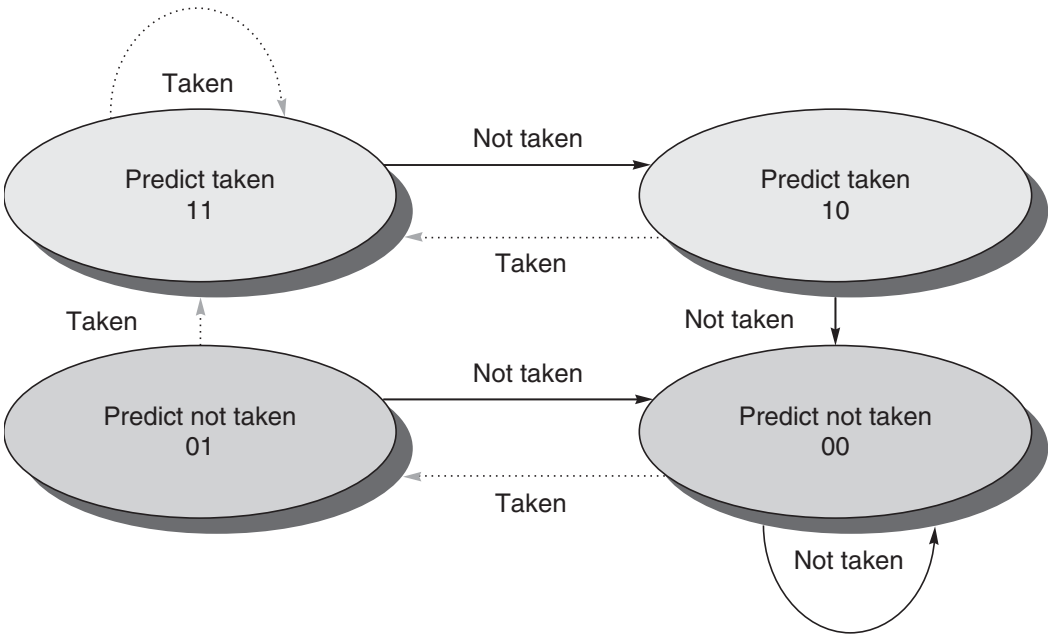


Figure 2: The 2-Bit Predictor Scheme (Taken from reference [1])

This paper studies the dependencies of a simple GShare 2-bit predictor by analyzing two trace files as the parameters of M and N are varied.

2 Branch Predictor Design

The branch predictor simulator was implemented using C++. A gshare class was written with 6 fields, GPB, RB, GHB, bpTable, correct, mispredict. GPB is variable storing the predictor bits in the PC address, M. RB is the variable storing the number of register bits for the global history buffer (also GHB in the code). The PHT is represented by a vector, bpTable, which holds a number $0-3$. The last two fields, correct and mispredict, store the number of correct and incorrect predictions, respectively.

The code takes in arguments <M><N><tracefile> and parses the arguments. The code then opens a file-stream. A gshare class is constructed and passed the initial arguments of GPB and RB (M and N). The file is then read line by line with the first string stored in `address` and the next character stored in `outcome`. The address and outcome are passed as arguments to the `branch_predict()` function. The address passed to `branch_predict()` is first passed to `parseAddress()` to remove the last 2 LSB's.

```
int main(int argc, char *argv[]) {
    //Implementing argument command line
    if (argc != 4) {
        std::cout<<"Proper Useage: "<< std::endl
        << argv[0]
        <<" <GPB>" << "<RV>"
        <<"<TRACE_FILE>"<<std::endl;
        exit (EXIT_FAILURE);
    }

    int GPB = atoi(argv[1]);
    int RB = atoi(argv[2]);
    std::string fileName = argv[3];

    std::ifstream myFile;
    myFile.open(fileName.c_str());
    char outcome;
    std::string address;

    gshare g(GPB, RB);

    // While the file still has lines, continue updating the cache
    if(myFile.is_open()) {
        while (myFile >> address >> outcome) {
            g.branch_predict(g.parseAddress(address), toupper(outcome));
        }
        myFile.close();
    } else {
        std::cout<<"The file <"<<fileName<<"> could not be opened!"<<std::endl;
        exit (EXIT_FAILURE);
    }

    std::cout<<GPB<<std::setw(10)<<RB<<std::setw(20);
    std::cout<<std::setprecision(4)<<g.get_mispredict_rate()<<std::setw(20)<<std::endl;

    return 0;
}
```

Once the file finishes being read, the mis-prediction rate is output to the screen. Further analysis of the functions in the gshare class is explained below.

```
void set_params(int GPB, int RB) {
    this->GPB = GPB;
    this->RB = RB;
    GHB = 0;
    correct = 0;
    mispredict = 0;

    // set size of history table
    long long unsigned int tableSize = (long long unsigned int)pow(2,GPB);
    for (int j = 0; j< tableSize; j++){
        bpTable.push_back(2);
```

```
    }
}
```

The function `set_params()` is called by the class constructor and initializes all variables of the class. The size of the PHT= 2^M. This is used to push new vector elements onto the table.

```
int branch_predict(long long unsigned int pc, char outcome) {
    // Pad on 0's to the same length of GPB
    long long unsigned int MGHB = GHB*(long long unsigned int)pow(2, GPB-RB);
    long long unsigned int MPC = pc%(long long unsigned int)pow(2, GPB);
    long long unsigned int index = MGHB ^ MPC;

    if (state(bpTable.at(index))==outcome) {
        correct++;
        bpTable.at(index)=increment(bpTable.at(index), outcome);
    } else {
        mispredict++;
        bpTable.at(index)=increment(bpTable.at(index), outcome);
    }
    updateGHB(outcome);
}
```

The function `branch_predict()` is called n times where n is the number of instructions in each trace file. The parsed address is taken in as pc. The GHB has $2^{\text{GPB}-\text{RB}}$ 0's appended to the end and stored in the modified GHB (MGHB). Take the remainder of the PC when divided by 2^{GPB} (which corresponds to the M bits) and store it in modified program counter (MPC). The index for the PHT is found by XORing the MPC and MGHB.

The conditional statements call the `state()` function which returns character value taken (T) or not taken (N) stored in the in the PHT and compare it to the outcome. This will update the corresponding correct or mispredict parameters and increment the values based on the 2-bit prediction scheme by storing the value returned from the `increment()` function. The GHB is the updated by `updateGHB()`.

```
void updateGHB (char outcome) {
    // First remove last bit (LRU state)
    GHB =GHB/2;
    // Clear MSB by performing bitwise AND operation with 011...1
    // If the path is not taken, then all is done, otherwise append 1
    // with bitwise OR operation of 100...0 and GHB.
    GHB = GHB & ((long long unsigned int)pow(2,RB -1), -1);
    if (outcome == 'N')
        return;
    GHB = GHB | (long long unsigned int)pow(2, RB - 1);
}
```

`updateGHB()` takes in the outcome of the branch. First the last bit is removed as with the FIFO policy by GHB by 2. A 0 is padded onto the beginning by performing a bitwise-AND (&) operation. The value is $2^{\text{RB}-1} - 1$ which creates a bit string of length RB with all bits set to 1 except for the MSB. This ensures all previous bits in the GHB remain the same as previously except now the

MSB is a 0. Therefore, if the outcome was not-taken then all work is done and the function returns. Otherwise, 1 is set as the MSB by taking 2^{RB-1} and bitwise-OR with GHB.

```
int increment(int bitState, char outcome) {
    if (outcome == 'T') {
        if(bitState == 3)
            return bitState;
        return bitState + 1;
    } else {
        if(bitState == 0)
            return bitState;
        return bitState - 1;
    }
}
```

The `increment()` function updates the PHT. It takes the previously stored prediction and the outcome. If the outcome was taken, we increment up, otherwise the value is decrement. If the value cannot be changed then the value is left alone.

3 Branch Prediction Accuracy

The performance for the GShare simulator is quantified by misprediction rate, that is the number of mispredictions over the total instructions. The calculation for the misprediction rate is as follows,

$$\text{misprediction rate} = \frac{\text{incorrect}}{\text{correct} + \text{incorrect}} \times 100\%$$

The studies were done with the "gobmk_trace.txt" and "mcf_trace.txt". The following studies were done to understand the prediction accuracy:

(A) For each of the provided traces/applications fix M at 4 bits and vary N to 1, 2, 3 and 4.

(B) For each of the provided traces/applications fix N at 4 bits (means you history of last 4 outcomes) and vary M to 4, 5, 6 and 7.

(C) For each of the provided traces, assume we have N to be 0 and M to be varied to 4, 5, 6 and 7.

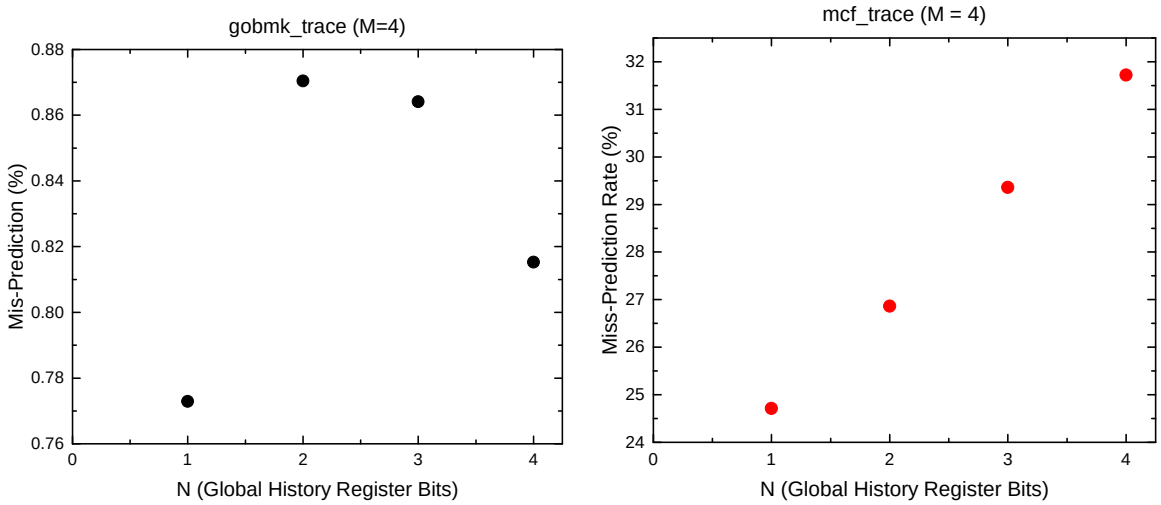
The results can be seen in Appendix A.

4 GShare Branch Predictor Analysis

The data from the runs were plotted using Origin Pro 2016 which can be seen below.

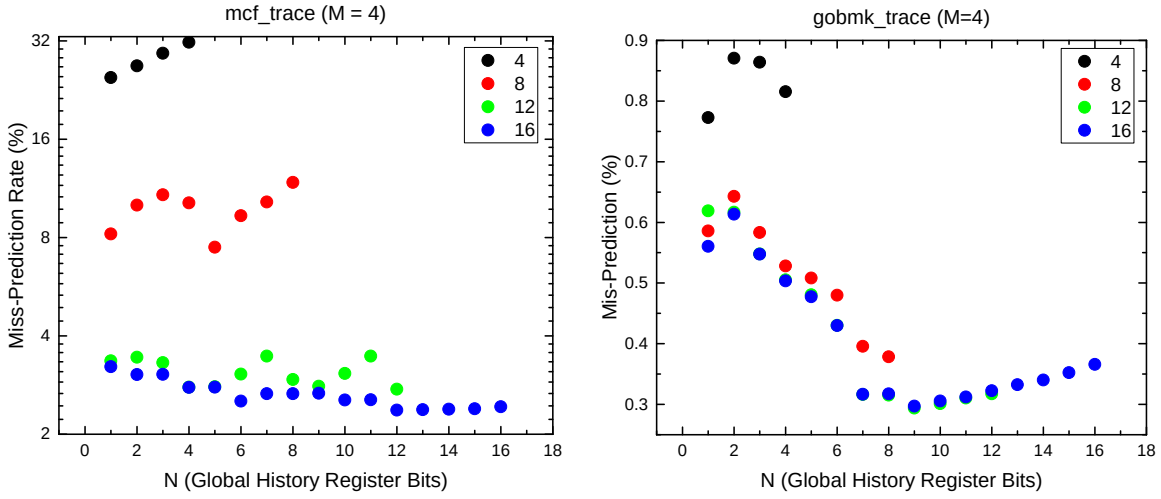
4.1 Part A

Figure 3 shows the misprediction rate for M =4 and as N is increased. The notable behavior between the two is the near linear relationship to M and N for mcf and the maximum for gobmk. There is not much else that can be obtained from this data so the program was re-run for additional values of M and varying N from 1 to M.



(a) (b)

Figure 3: The mispredicton rate for M = 4 and N is varied from 1 to 4



(a) (b)

Figure 4: The mispredicton rate for M = 4,8,12,16 and N is from 1 to M

Figure 4 displays the results for varying M and N.I M was selected to be 4, 8, 12, and 16. One notable trend is the overall decrease in the misprediction rate as M is increased by 4. Figure 4 (b) uses a $\log_2$ scale for the ordinate. It may be noted that the change is prediction rate increases greatly

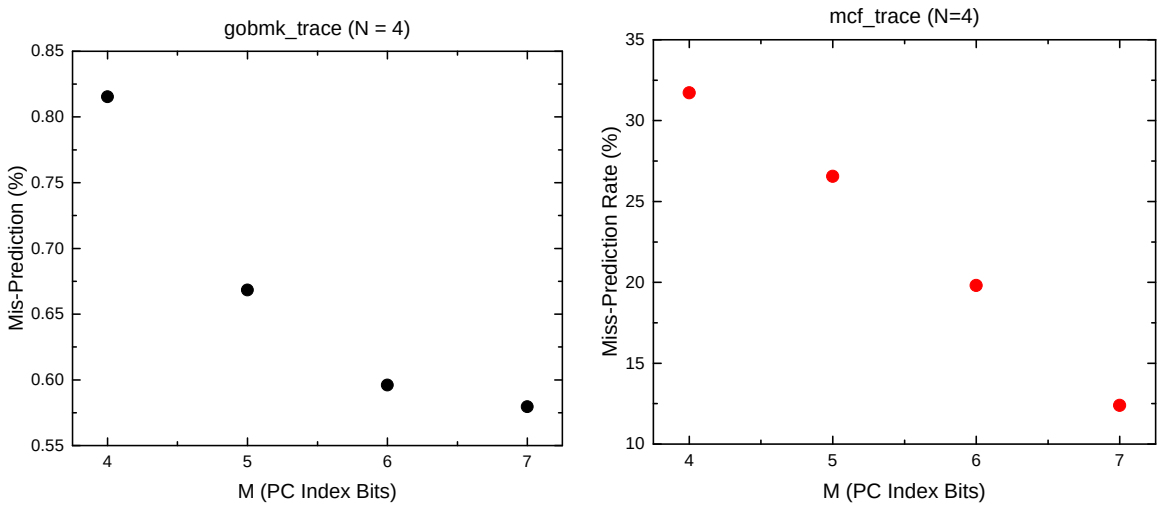
as M is doubled. Other behavior shows that as N increases (the size of the GHB) the misprediction ratio decreases for both for large M.

For small M (lower than 8 bits), the misprediction ratio is high and increases for both as N increases when M is 4. This is because GShare works well with large page tables. One important observation is the dramatic increase in the misprediction rate for the gobmk file when M = 16. When $N \geq 10$, the misprediction rate increases linearly. This may be attributed to program the trace file delineates. It implies that dependency is lost after 9 instructions which will throw off the prediction history.

Another key observation is the sheer difference in magnitude of the misprediction rates of both files. The mcf file more than 32 larger for the M = 4 case. This may mean that the gobmk has a greater global dependency than the mcf file. Then the gobmk file would benefit more from the GShare implementation.

4.2 Part B

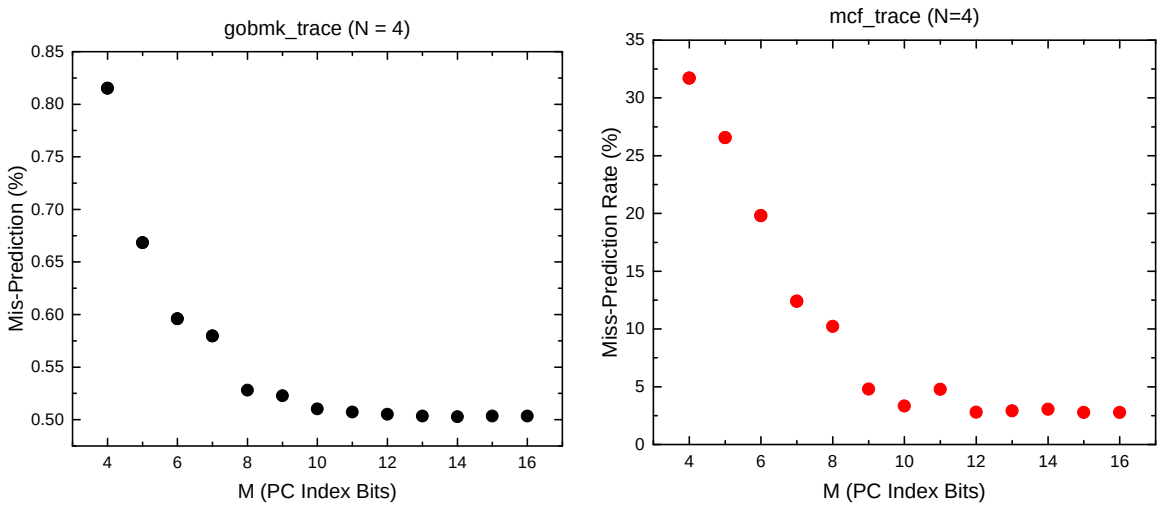
The following figures display the dependency on the size of the PHT when N = 4. It may be observed that for both files, as M increases, the mis-prediction rate decreases. With the 2^M size dependency for the PHT, it should be expected that there is an exponential dependency between the mis-prediction rate and M.



(a) (b)

Figure 5: The mispredicton rate for N = 4 and M is from N to 7

By extending the data to 16 bits for M, the trend may be observed much more clearly. Notice that in Figure 6 (a), the exponential curve is much more relaxed than that of Figure 6 (b). The mcf trace file sees almost a linear dependency until 10 bits. The difference in behavior may be explained by the difference in the intial mis-prediction rates. The mcf file has a much larger mis-prediction rate than the gobmk file; therefore the effect of increasing the PHT size would be felt more in the file where accuracy can be greatly improved.

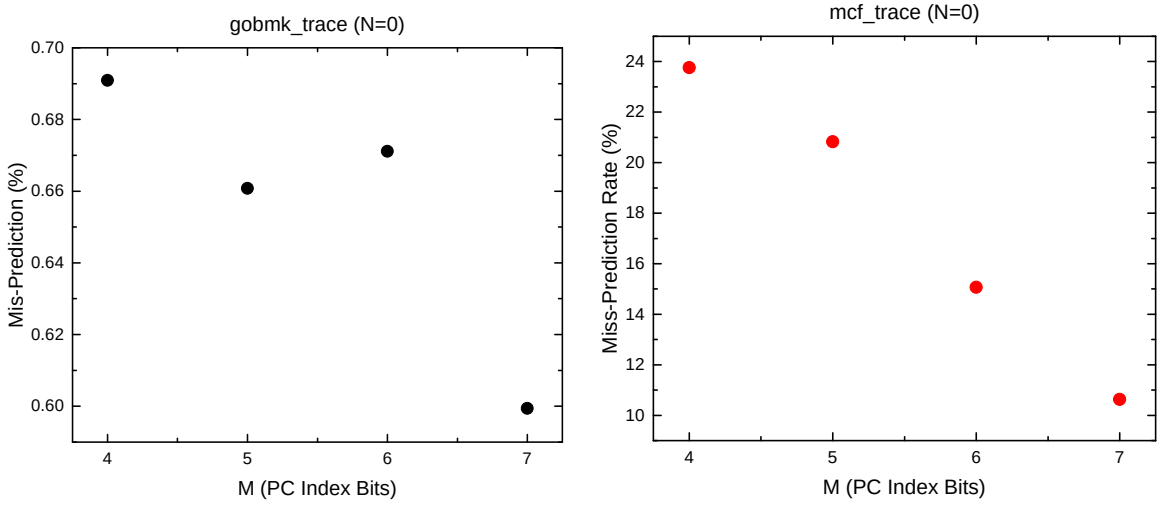


(a) (b)

Figure 6: The mispredicton rate for N = 4 and M is from N to 16

4.3 Part C

The figures below describe mis-prediction rate when N = 0, that is, when there is no GHB. It is apparent that for Figure 7, the mis-prediction rate is smaller than compared to when N = 4. This may be due to the size of the PHT table since GShare is good for when M is large. With M being so small, more PC addresses map to similar indexes, consequently interfering with the predictions for other branches.

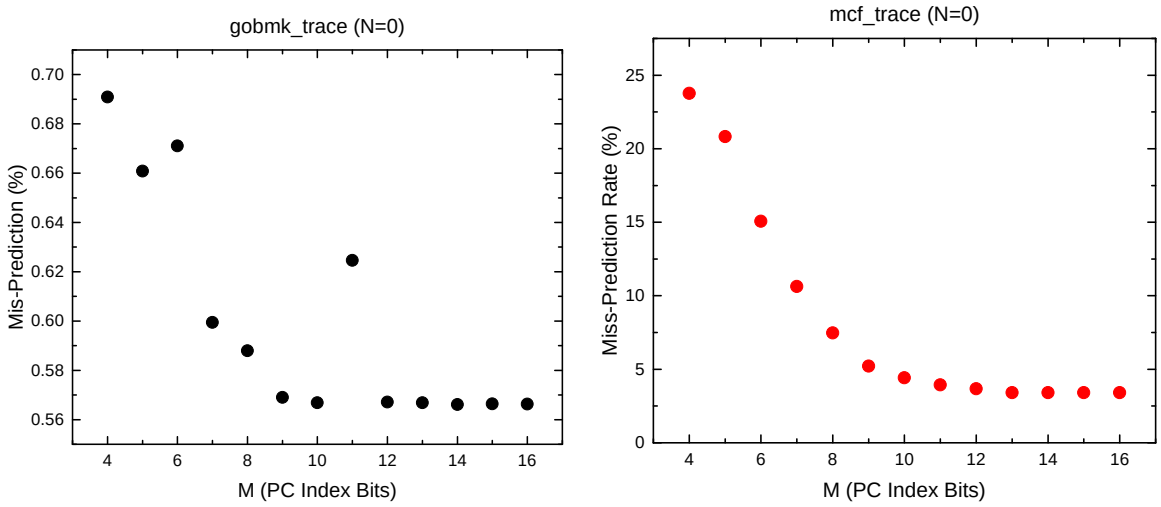


(a) (b)

Figure 7: The mispredicton rate for N = 0 and M is from 4 to 7

In Figure 8, the data is extended and is notable that the variability of the data for the gobmk file

is much greater than when N = 4. Also, for Figure 8 (a), the mis-prediction rate approach 0.56% as opposed to 0.50% for N = 4 in Figure 7 (a). The mis-prediction rate has a smaller derivative for N = 0 in Figure 8 (b) than in Figure 7 (b). It seems that the rate of change in Figure 8 (b) has a maximum in the vicinity of 6 bits. This demonstrates that N = 0 reduces the prediction rate for large M but seems better for smaller M.



(a) (b)

Figure 8: The mispredicton rate for N = 0 and M is from 4 to 16

5 Summary

Studying the effects of GHB size and PHT size showed significant differences on the performance of branch prediction for the GShare predictor. The mis-prediction rate has an exponential dependency on the size of the PHT. However, the effectiveness of the PHT is dependent upon the size of the GHB. For smaller GHB, a smaller PHT table would suffice, but as the table size increases, the GHB size needs to increase.

Appendix A

A.1 gobmk_trace

Table 1: M=4, N is varied 1 through 4

M	N	Miss-prediction Ratio
4	1	0.7729
4	2	0.8704
4	3	0.8641
4	4	0.8153

Table 2: N = 4, M is swept from N to 16

M (bits)	N (bits)	Miss-Prediction Rate (%)
4	4	0.8153
5	4	0.6683
6	4	0.596
7	4	0.5796
8	4	0.5281
9	4	0.5227
10	4	0.5103
11	4	0.5073
12	4	0.5052
13	4	0.5034
14	4	0.5029
15	4	0.5035

Table 3: N = 4, M is swept from N to 16

M (bits)	N (bits)	Miss-Prediction Rate (%)
4	0	0.6909
5	0	0.6608
6	0	0.6711
7	0	0.5994
8	0	0.5879
9	0	0.569
10	0	0.5668
11	0	0.6246
12	0	0.5671
13	0	0.5668
14	0	0.5661
15	0	0.5664

A.2 mcf_trace

Table 4: M=4, N is varied 1 through 4

M	N	Miss-prediction Ratio
4	1	24.71
4	2	26.86
4	3	29.36
4	4	31.72

Table 5: N = 4, M is swept from N to 16

M (bits)	N (bits)	Miss-Prediction Rate (%)
4	4	31.72
5	4	26.56
6	4	19.81
7	4	12.39
8	4	10.22
9	4	4.788
10	4	3.32
11	4	4.765
12	4	2.786
13	4	2.914
14	4	3.045
15	4	2.782

Table 6: N = 4, M is swept from N to 16

M (bits)	N (bits)	Miss-Prediction Rate (%)
4	0	23.76
5	0	20.83
6	0	15.07
7	0	10.63
8	0	7.469
9	0	5.21
10	0	4.418
11	0	3.932
12	0	3.667
13	0	3.403
14	0	3.403
15	0	3.403

References

[1] Hennessy, John L., Patterson, David A., *Computer Architecture: A quantitative Approach*, Massachusetts: Morgan Kaufmann, (2012).

[2] Lin, W., Madhavaram, R., & An-Yi, Y. Improving Branch Prediction Performance with A Generalized Design for Dynamic Branch Predictors. *Informatica* (03505596), **29**(2), 365 - 373 (2005).