Project 2 Report
EEL4768

Yousef Awad

November 2025

Contents

Part A: Varying N (M=4) **2**

Part B: Varying M (N=4) **3**

Part C: Varying M (N=0) **4**

Comparison of Part B and Part C **4**
Analysis 4
Overlay Graphs 5

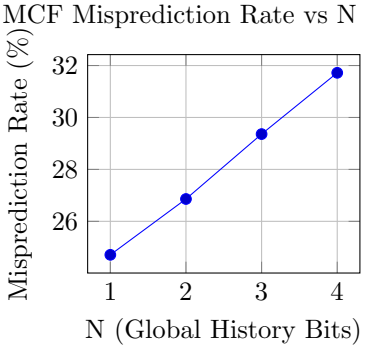
Source Code (sim.c) **6**

Part A: Varying N (M=4)

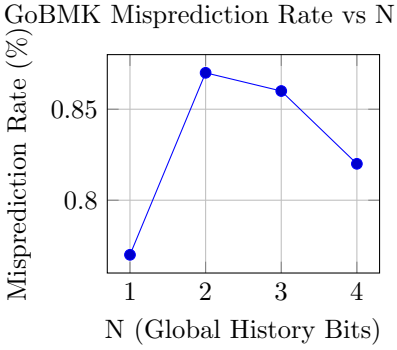
In this part, we fix M at 4 bits (16 entries) and vary N from 1 to 4.

N	MCF	GoBMK
1	24.71	0.77
2	26.86	0.87
3	29.36	0.86
4	31.72	0.82

MCF Misprediction Rate vs N (M=4)



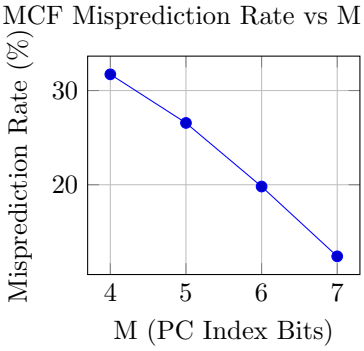
GoBMK Misprediction Rate vs N (M=4)

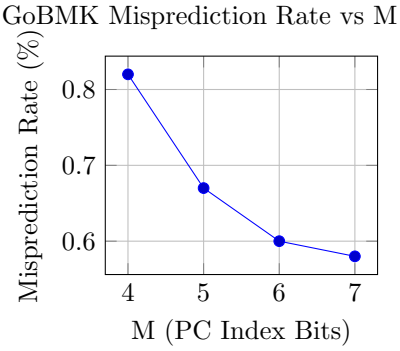


Part B: Varying M (N=4)

In this part, we fix N at 4 bits and vary M from 4 to 7.

M	MCF	GoBMK
4	31.72	0.82
5	26.56	0.67
6	19.81	0.6
7	12.4	0.58

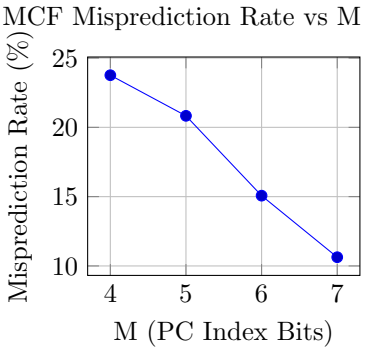


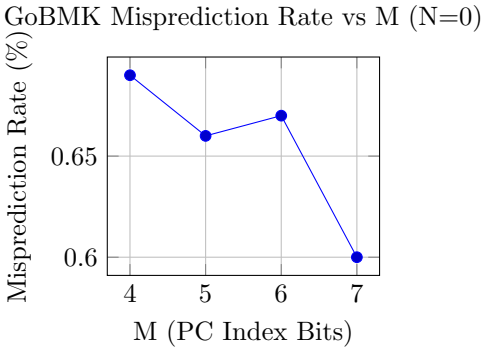


Part C: Varying M (N=0)

In this part, we fix N at 0 bits (no global history) and vary M from 4 to 7. This effectively makes it a bimodal predictor.

M	MCF	GoBMK
4	23.76	0.69
5	20.83	0.66
6	15.07	0.67
7	10.63	0.6





Comparison of Part B and Part C

In Part B, we used a Gshare predictor with a fixed global history length of $N = 4$ while varying the table size index bits M. In Part C, we used a Bimodal predictor ($N = 0$), effectively using only the PC to index the table, while varying M.

Analysis

MCF Trace: The Bimodal predictor (Part C, $N = 0$) consistently outperforms the Gshare predictor (Part B, $N = 4$) across all tested table sizes ($M = 4$ to 7).

- At $M = 4$, the difference is significant: Gshare has a misprediction rate of 31.72% while Bimodal is 23.76%.
- As M increases, the gap narrows. At $M = 7$, Gshare is 12.40% and Bimodal is 10.63%.

Reasoning: The MCF trace likely suffers from significant destructive aliasing (interference) in the Pattern History Table when using Gshare with small table sizes. The XOR hashing of the PC and GHR maps multiple unrelated branches to the same entry, causing them to overwrite each other's state. Since the table sizes are very small (e.g., $2^4 = 16$ entries), this interference is severe. The

Bimodal predictor ($N = 0$) uses only the PC bits, which might provide a cleaner separation for the dominant branches in this specific trace at these small sizes.

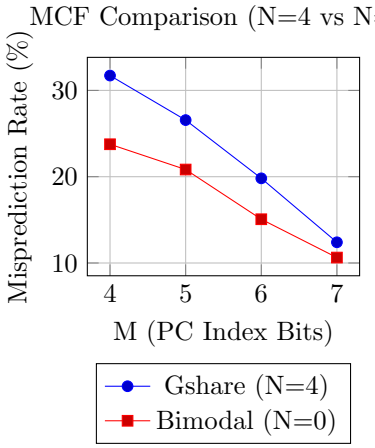
GoBMK Trace: The results for GoBMK show a crossover point where Gshare begins to outperform Bimodal.

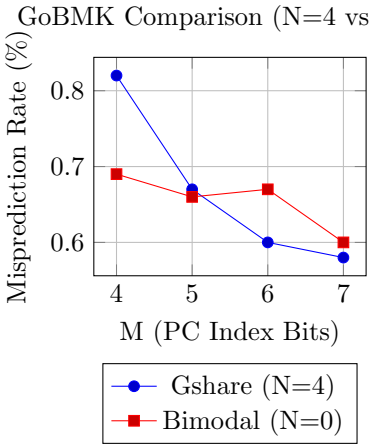
- At small table sizes ($M = 4, 5$), Bimodal ($N = 0$) is slightly better or comparable to Gshare ($N = 4$).
- At larger table sizes ($M = 6, 7$), Gshare starts to outperform Bimodal. For instance, at $M = 7$, Gshare achieves 0.58% compared to Bimodal's 0.60%.

Reasoning: GoBMK appears to benefit from global history information. However, at very small table sizes ($M = 4, 5$), the aliasing penalty of Gshare outweighs the benefit of the history. As the table size increases ($M = 6, 7$), the interference is reduced enough that the predictive power of the global history becomes advantageous, allowing Gshare to overtake the simple Bimodal predictor.

Overlay Graphs

The following graphs overlay the results from Part B (Gshare, $N = 4$) and Part C (Bimodal, $N = 0$) to visualize the differences.





Source Code (sim.c)

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

typedef struct
{
    uint8_t *pht;
    uint32_t ghr;
    int M;
    int N;
} Predictor;

int init_predictor(Predictor *pred, int m, int n)
{
    if (m < 0 || m > 30) // Prevent excessive allocation or
    negative shifts
    {
        return 0;
    }
    if (n < 0 || n > m) // N must be <= M
    {
        return 0;
    }

    pred->M = m;
    pred->N = n;
    size_t pht_size = 1U << pred->M;
    pred->pht = (uint8_t *)malloc(pht_size * sizeof(uint8_t));

    if (pred->pht == NULL)
    {
        return 0; // Return 0 if memory allocation fails
    }

    // Initialize to Weakly Taken
    memset(pred->pht, 2, pht_size);
    // GHR is set to 0
    pred->ghr = 0;
    return 1;
}

int main(int argc, char *argv[])
{
    if (argc != 5)
    {
        printf("Usage: %s gshare <GPB> <RB> <Trace_File>\n", argv
    [0]);
        printf("1) GPB: The number of PC bits used to index the
    gshare table.\n");
        printf("2) RB: The global history register bits used to
    index the gshare table.\n");
        printf("3) Trace_File: The trace file name along with its
    extension.\n");
        return 1;
```

```
    }

    if (strcmp(argv[1], "gshare") != 0)
    {
        printf("Error: Only 'gshare' predictor is supported. Usage:
     %s gshare <GPB> <RB> <Trace_File>\n", argv[0]);
        return 1;
    }

    char *endptr;
    long m_long = strtol(argv[2], &endptr, 10);
    if (*endptr != '\0' || m_long < 0 || m_long > 30)
    {
      printf("Error: Invalid or out-of-range value for GPB (M): %s\
    n", argv[2]);
      return 1;
    }
    // M was safely parsed
    int m = (int)m_long;

    long n_long = strtol(argv[3], &endptr, 10);
    if (*endptr != '\0' || n_long < 0 || n_long > m)
    {
      printf("Error: Invalid or out-of-range value for RB (N): %s\n
    ", argv[3]);
      return 1;
    }
    // N was safely parsed
    int n = (int)n_long;

    char *trace_file = argv[4];

    Predictor pred;
    pred.pht = NULL; // Ensuring pointer is null and not garbage
    value.
    if (!init_predictor(&pred, m, n))
    {
        // Return 0 if initialization fails
        printf("Error: Invalid parameters or memory allocation
    failure (M=%d, N=%d)\n", m, n);
        return 1;
    }

    unsigned long long total_branches = 0;
    unsigned long long mispredictions = 0;

    FILE *fp = fopen(trace_file, "r");
    if (fp == NULL)
    {
        // Return 0 if file opening fails
        printf("Error opening file %s\n", trace_file);
        free(pred.pht);
        return 1;
    }

    char line[256];
    // Read each line of the trace file
```

```
    while (fgets(line, sizeof(line), fp))
    {
        unsigned int pc;
        char outcome_char;
        char outcome_str[10];

        if (sscanf(line, "%x %9s", &pc, outcome_str) != 2)
        {
            // Skip invalid lines
            continue;
        }
        outcome_char = outcome_str[0];

        int actual_taken = (outcome_char == 't');

        // Index Calculation
        // PC index: bits M+1 to 2
        // We need to mask PC to get the relevant bits.
        // But simply shifting right by 2 and masking with (1<<M)-1 gives the M bits.
        unsigned int pc_index = (pc >> 2) & ((1U << pred.M) - 1);

        // Index Calculation:
        // XOR the N-bit GHR with the upper N bits of the M-bit PC index.
        unsigned int index = pc_index ^ (pred.ghr << (pred.M - pred.N));

        // Prediction
        int counter = pred.pht[index];
        int prediction_taken = (counter >= 2);

        if (prediction_taken != actual_taken)
        {
            // Increment mispredictions if prediction is incorrect
            mispredictions++;
        }

        total_branches++;

        // Update PHT
        if (actual_taken)
        {
            // Increment counter if prediction is correct
            if (counter < 3)
            {
                pred.pht[index]++;
            }
        }
        else
        {
            // Decrement counter if prediction is incorrect
            if (counter > 0)
            {
                pred.pht[index]--;
            }
        }
```

```
157
158         // Update GHR
159         if (pred.N > 0)
160         {
161             // Shift right 1, insert new outcome at MSB (bit N-1).
162             pred.ghr = (pred.ghr >> 1) | (actual_taken << (pred.N -
     1));
163         }
164     }
165
166     // Close the trace file
167     fclose(fp);
168
169     // Print the misprediction rate
170     printf("GBP=%d, RB=%d\n", m, n);
171     printf("Misprediction Rate: %.2f%%\n", (double)mispredictions /
     total_branches * 100.0);
172
173     // Free the PHT
174     free(pred.pht);
175
176     // Returning 0 since the program executed successfully
177     return 0;
178 }
```