Simulation of Branch Predictor

Machine Problem 2 – Project Report

Pratheek Bramhasamudra Mallikarjuna

pbramha@ncsu.edu

200065594

I, Pratheek Bramhasamudra Mallikarjuna, hereby declare that, I have complied with all the requirements necessary for maintaining academic integrity to the best of my knowledge while doing this project.

Pratheek Bramhasamudra Mallikarjuna

Table of Contents

1. INTRODUCTION

While programming, programmers use many coding techniques, which include branching codes like if-else, switch, and loops like for, while, along with functions to enforce modularity. These codes, when converted into machine instructions, will be branch instructions. Generally, with pipeline architecture of the processors, these branch instructions cause stall cycles, because we will not know if we are supposed to branch or not, until the branch instructions are executed, hence hindering the pipeline performance. Other this is, we will not know which the location of the branch as well.

To deal with these issues, we can come up with the branch predictor logic, which essentially predicts the possible outcome of the branch. For this we are using Bimodal, Gshare and Hybrid branch predictors. To handle the branch target location, we use Branch Target Buffer (BTB), which is essentially a cache with branch targets.

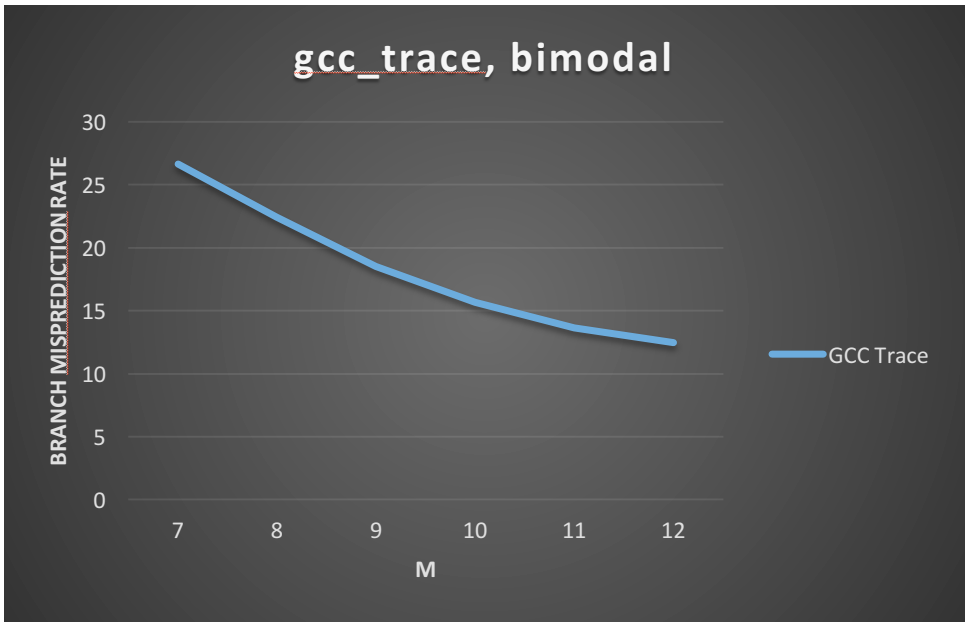
The project is implemented in C++, with Predictor as a class. This is used for both Bimodal and Gshare. It is also used as a select algorithm between bimodal and gshare in case of Hybrid predictor.

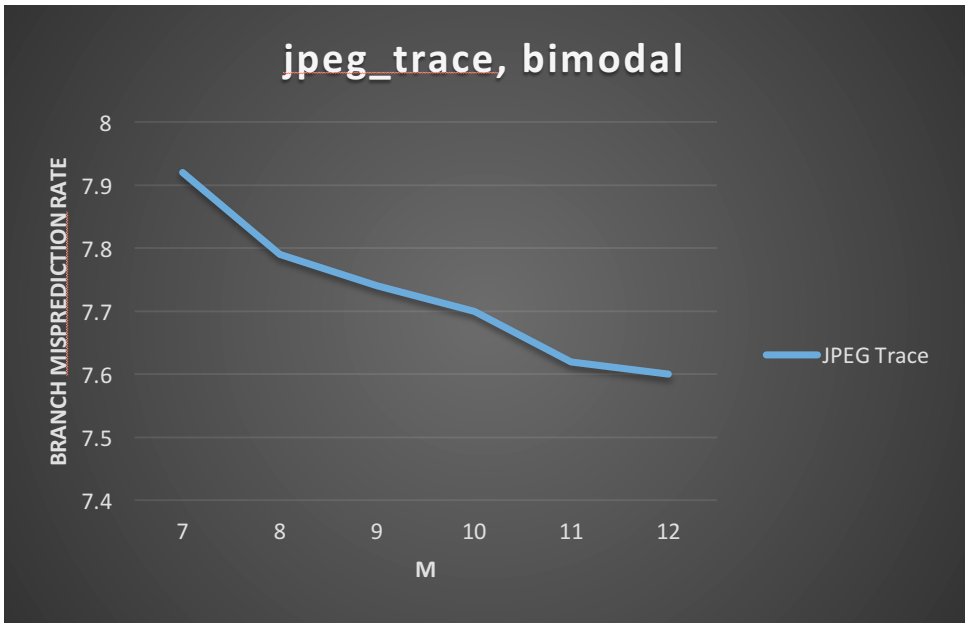
BTB used L1 cache with block size as 4 since we are ignoring the last 2 bits of PC for indexing, as they are 0.

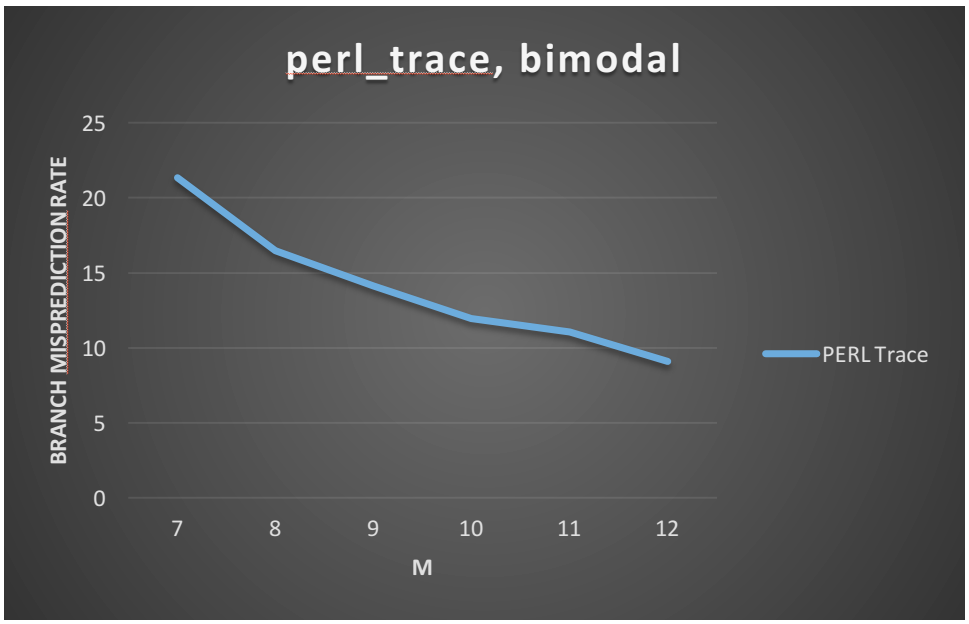
2. RESULTS

2.1 BIMODAL

Following graphs show the trend of misprediction rate with respect to number of PC bits used to index the bimodal predictor table.







2.2 Analysis: Bimodal

We can see that, with increase in M, the mispredictions rate reduces for all the traces. However, the rate of decrease is different for different traces. While GCC and PERL start with pretty high misprediction rate for lower value of M, JPEG starts with relatively low value of misprediction rate and stays relatively same. Both GCC and PERL show big improvements with increase in M, but tend to slowly saturate at around M = 10, 11. Both PERL and JPEG go as low as 7-8%, while GCC settles for much higher value of around 12%.

2.3 Design: Bimodal

Design recommendations are done in the following sections.

2.3.1 GCC

We can see that, the misprediction rates fall rapidly initially with increase in M, wile tend to saturate with higher values of M. With M increasing from 11 to 12, we only see a misprediction drop of around 1%, while the size required increase is double. The trend is same with M increasing from 10 to 11, where misprediction decreases by around 2% but size required increases doubles. The decrease in misprediction rate is higher for lower M values. By this we can conclude that, M = 10, gives fairly good prediction with lower size requirements

2.3.2 JPEG

For this trace, the trend is almost flat graph, where misprediction rates are almost around 7%. For M = 7, we see the misprediction rate is around 8%, while for M = 12, it is 7.6%. This is a decrease of 0.4 %, for increase in size requirement by 32 times. Taking this

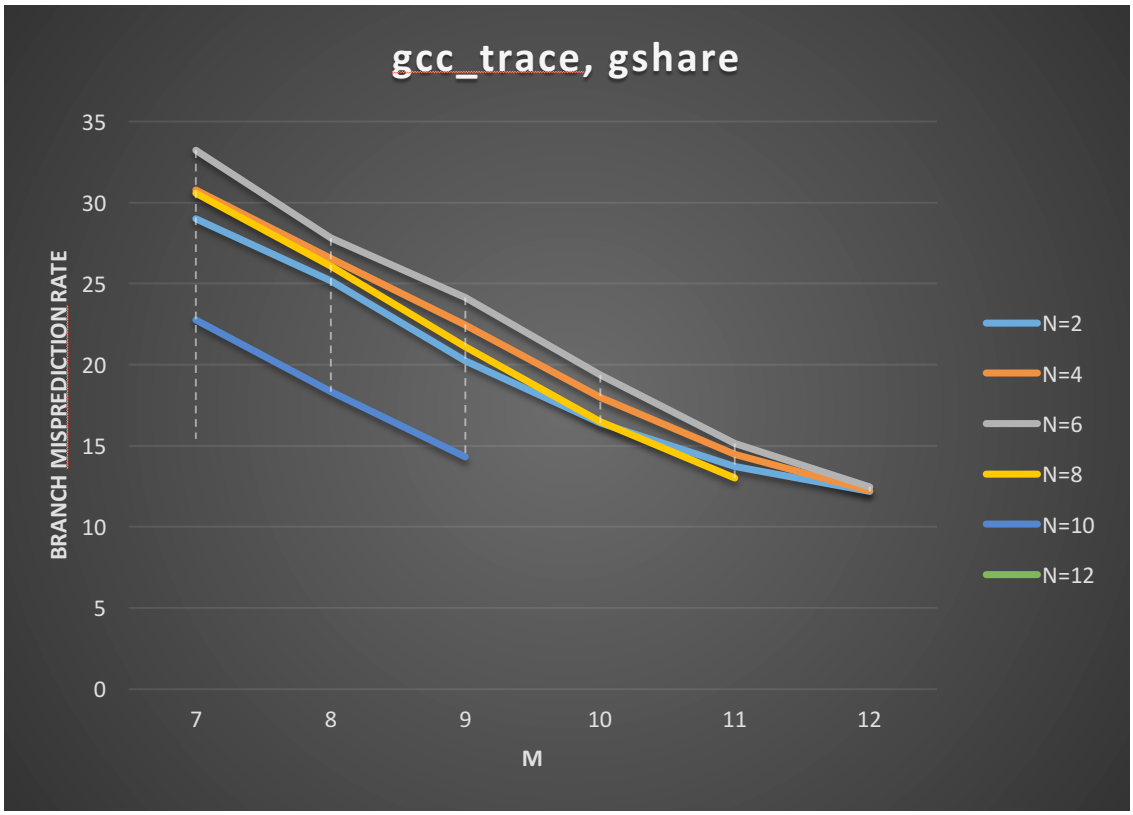
into consideration, we can manage fair amount of prediction for M = 7, 8 with lower memory size.

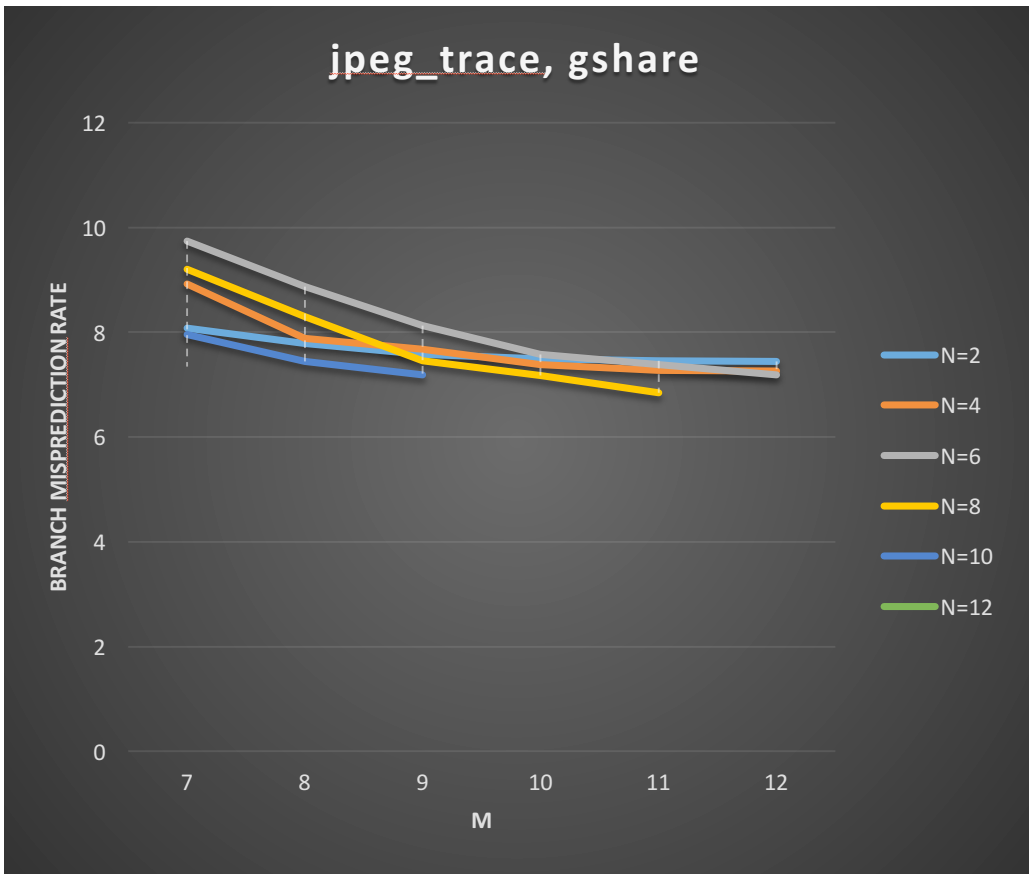
2.3.3 PERL

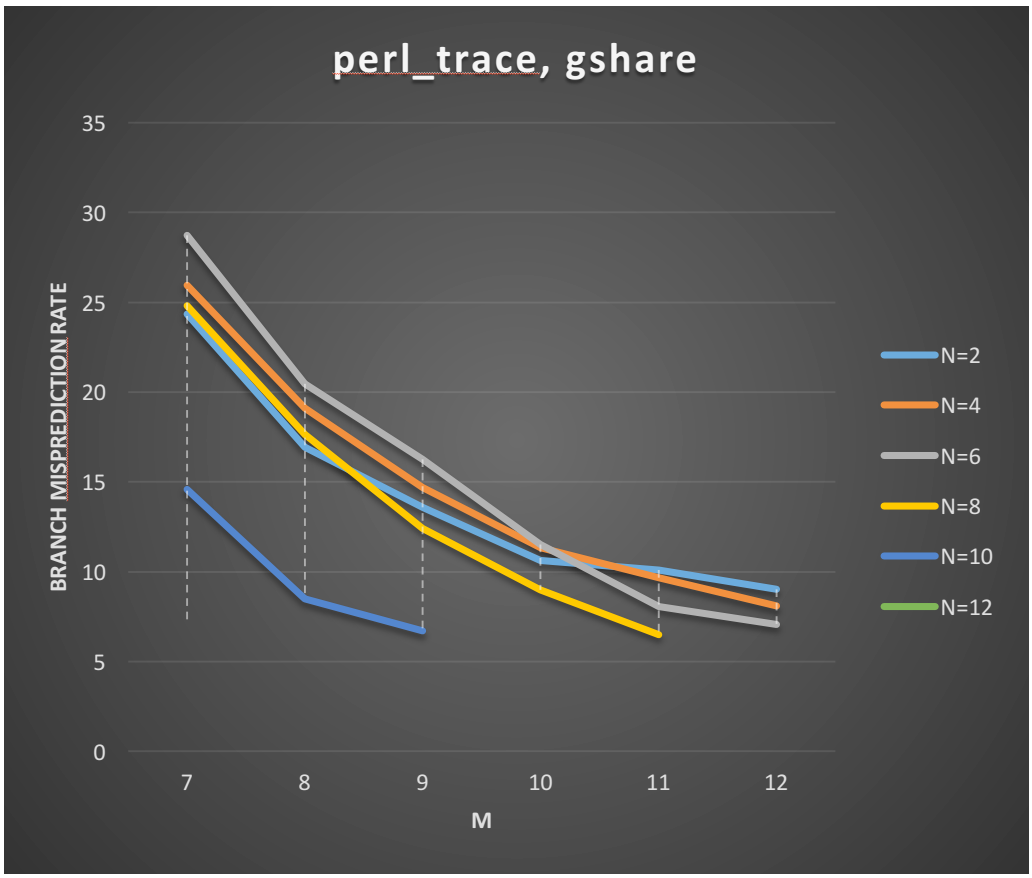
The trend is similar to GCC trace, where the drop in misprediction is higher when the M is small and as M increases the drop saturates. In this case, similar to GCC, M = 10, provides a fair prediction with misprediction of around 12% with very less memory size required compared to say M = 12.

2.4 GSHARE

Following are the figures, which show the trend of mispredictions with change in number of bits used to index PC and number of branch history register bits.




jpeg_trace, gshare
BRANCH MISPREDICTION RATE
12
10
8
6
4
2
0
7
8
9
10
11
12
M
N=2
N=4
N=6
N=8
N=10
N=12



2.5 Analysis: Gshare

In all the graphs, the general trend we is see is, improved prediction rate with increase in M, that is, prediction table index bits. Another trend we can see is that, with increase in M, the improvements gained tend to saturate, in other words, we see the diminishing returns. Both GCC and PERL start with higher misprediction rates for M = 7 and reduce with increase in M, while JPEG starts with around 9% misprediction and stays relatively flat. While the PERL reaches as low as 7-8% misprediction with M = 11, 12, GCC settles for 12 – 13% misprediction.

On the other hand, effect of increasing N is different. Increasing N may not have a good effect in the prediction as we can see that, N = 2 gives better results than N = 3, 4. Additionally, N = 11, seems to give the best results for all the traces, while N = 10 gives good result as well.

2.6 Design: Gshare

2.6.1 GCC

Since history register is a small register, in crease in N will not have a bigger effect on the size constrains. So, we can choose a higher N value of 10 or 11.

As seen by the graphs, the saturation starts at around M = 10, where the size requirements doubles with a small decrease in misprediction rate. Hence we can go with M = 10 for this trace.

2.6.2 JPEG

Similar to GCC, we can choose, N = 10 or 11 for history register. However, since the performance is almost the same for all the M values, we can go with lesser M, say, M = 8, 9, to obtain a fairly good prediction rate for this trace.

2.6.3 PERL

The trends are similar to GCC. WE can choose history register size to be N = 10,11. Similar to GCC, here, the performance gains saturates at M = 10. So, we can choose M = 10, for fairly good performance with lesser requirements for memory.