



Term Project Final Report

Pipeline Branch Prediction Strategies

[Team Name: IFElse] (Bogki Yun, Yuguang Qiu, Qian Mai)

CSCI 5593 Advanced Computer Architecture
Computer Science and Engineering
University of Colorado Denver

Contents

1. Abstract.....	3
2. Motivation and Problem.....	4
3. Background knowledge and Introduction.....	4
3.1 Pipelining	4
3.2 Branch Instruction and Branch Table.....	6
3.3 Branch Prediction Classification.....	7
3.4 Branch Prediction & Branch Predictor	8
4. Solutions.....	9
4.1 Bimodal Predictor Calculation	9
4.2 G-share Predictor Calculation.....	9
4.3 Hybrid Predictor Calculation	11
5. Implementation.....	12
5.1 Compiling and Running Simulator	12
5.2 Command line.....	12
5.3 Develop Environment	14
5.4 Source and Compiling Example.....	15
5.5 Simulator Implement Flowchart.....	15
5.6 Trace and Trace File Format.....	16
5.7 Results Format	17
6. Evaluation and Results.....	18
6.1 The Parameters in the Cache Simulation	18
6.2 Trace1: gcc_trace.txt	18
6.3 Trace2: jpeg_trace.txt.....	21
6.4 Trace3: perl_trace.txt	23
6.5 Combined results for bimodal predictor.....	25
6.6 Combined results for G-share predictor	26

1. Abstract

High performance has become a major concern in the Computer Industry field. We all know that Pipelining is the major organizational technique that computers use to achieve high performance. It is a set of data processing elements connected in series, where the output of one element is the input of the next one. Because the processor works on different steps of the instruction at the same time, more instructions can be executed in a shorter period of time.

Every coin has two sides, a pipeline uniprocessor can run at a rate that is limited by its slowest stage and increases instruction throughput by performing multiple operations in parallel, however, does not reduce instruction latency. In high-performance computer systems, performance losses are usually due to conditional branch instructions, which perform a test by evaluating a logical condition and depending on the outcome of the condition that modify the program counter to take the branch or continue to the next instruction. In simple words, the performance might be dropped dramatically due to the processor must go through all steps to complete a single instruction from start to finish, meanwhile, wasting the system time by a branch instruction interruption. ^[1]

This can be minimized by predicting a branch outcome and fetching, decoding, and/or issuing subsequent instructions before the actual outcome is known. Therefore, Branch prediction has become an area of interest due to its effect on the performance of pipelined and superscalar processors. Various methods have been proposed to speculate the path of an instruction stream after a branch. Nevertheless, if branches are predicted poorly, it may create more work for the processor, such as flushing from the pipeline the incorrect code path that has begun execution before resuming execution at the correct location. Obviously, this problem leads to a scenario which absolutely is contrary to the original intention.

Based on this face, our project is aimed to achieve two goals. First of all, Study *branch predictions and related knowledge*. Implements a simulator of branch prediction strategies with the goal of finding a method to maximize the rate of correct predict. Evaluate our ideas by comparing the performance of several existing prediction schemes and find some efficient methods to improve the branch prediction strategies.



2. Motivation and Problem

Generally speaking, Branch prediction is used to overcome the fetch limitation imposed by control hazards in order to expose instruction-level parallelism (ILP). Branch instruction constitutes 15 to 30% of instruction executed on a typical machine. It can also break the smooth flow of instruction fetching and execution.

We know the key part of pipelined and superscalar system architectures that mask instruction execution latencies by exploiting (ILP). Branch prediction can be thought of as a sophisticated form of pre-fetch or a limited form of data prediction that attempts to predict the result of branch instructions so that a processor can speculatively fetch across basic-block boundaries.

Therefore, once the predictor has made a decision, the processor can execute instructions that depending on the outcome of the branch. If the branch prediction accuracy is high enough, this might be used to offset mis-prediction penalties. Obviously, the processor will likely have a better overall performance then.

Many branch prediction schemes have been proposed in the past decades. Our work is to build a powerful simulator and use selected benchmark programs to compare several well-known branch prediction schemes, which are biomodal, G-share and hybrid predictors.

3. Background knowledge and Introduction

3.1 Pipelining

The concept of Pipelining plays the significant rules in our project. A useful method of demonstration is like below. To the right is a generic pipeline with four stages: **1.Fetch 2. Decode 3. Execute 4. Write-back**

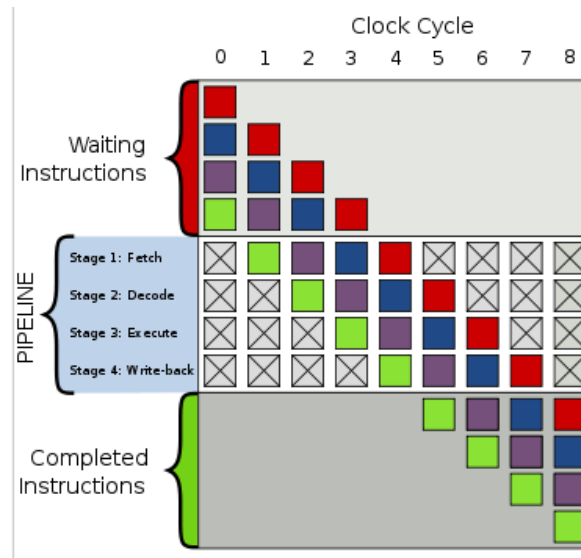


Figure 3.1.1: 4-stage pipeline

In this figure, the top gray box is the list of instructions waiting to be executed; the bottom gray box is the list of instructions that have been completed; and the middle white box is the pipeline^[2]

Time	Execution
0	Four instructions are waiting to be executed
1	<ul style="list-style-type: none"> The green instruction is fetched from memory
2	<ul style="list-style-type: none"> The green instruction is decoded The purple instruction is fetched from memory
3	<ul style="list-style-type: none"> The green instruction is executed (actual operation is performed) The purple instruction is decoded The blue instruction is fetched
4	<ul style="list-style-type: none"> The green instruction's results are written back to the register file or memory The purple instruction is executed The blue instruction is decoded The red instruction is fetched
5	<ul style="list-style-type: none"> The green instruction is completed The purple instruction is written back The blue instruction is executed The red instruction is decoded
6	<ul style="list-style-type: none"> The purple instruction is completed The blue instruction is written back The red instruction is executed
7	<ul style="list-style-type: none"> The blue instruction is completed The red instruction is written back
8	<ul style="list-style-type: none"> The red instruction is completed

Figure 3.1.2: Example of pipelining execution

3.2 Branch Instruction and Branch Table

A branch instruction can be either an unconditional branch, which always results in branching, or a conditional branch, which may or may not cause branching depending on some condition. Branch instructions are always relative to the current program counter. That is, the next instruction is obtained by adding a signed offset to current program counter, thus we get this equation.

$$PC += (\text{int}) \text{ offset}$$

In addition, branches are inherently re-locatable.

An unconditional branch instruction causes the Program location counter (PSW) to be set to the address specified in the register or the register plus a 12-bit offset, or the register & offset plus the value of an additional "index" register. The branch does not occur, and is treated as a no-op, for a BR instruction using register 0. The branch does not occur, and is treated as an instruction to load the address of the following instruction into the left register, if the right register in a BALR instruction is 0. No conditional branch - including unconditional branch - will occur if the index register is 0. These instructions may be one of the following types ^[3]:

- *Register to register (RR)*
- *Storage (RS)*
- *Indexed (RX)*

A conditional branch instruction causes the location counter in the PSW to be set to the address specified in the register or the register plus a 12-bit offset, if a condition is satisfied (and the register is not 0) and thus may be one of the following types:

- *Storage (RS)*
- *Indexed (RX)*
- *Register to register (RR)*

A branch table is a literally a set of contiguous unconditional branch instructions which are of equal length (usually 4 bytes), that are used to efficiently branch directly to one of this set, using an index. This index is often generated from some source input value that may itself be non-sequential as in the example below. This method is considerably faster than using either a binary search or sequential table lookup for example. Lookups involve compare instructions and a subsequent conditional branch.

3.3 Branch Prediction Classification

3.3.1 Static Branch Prediction

- The direction of each branch is predicted before a programs runs using either compile time or profiling.
- Static Branch prediction policies of "always predict ***taken***" and "always predict ***not taken***" - ***Average Mis-prediction = untaken branch frequency = 34% SPEC***

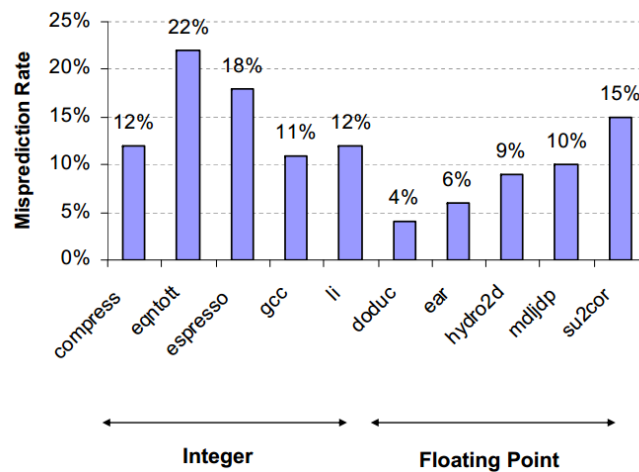


Figure 3.3.1: A typical static branch predictor performance

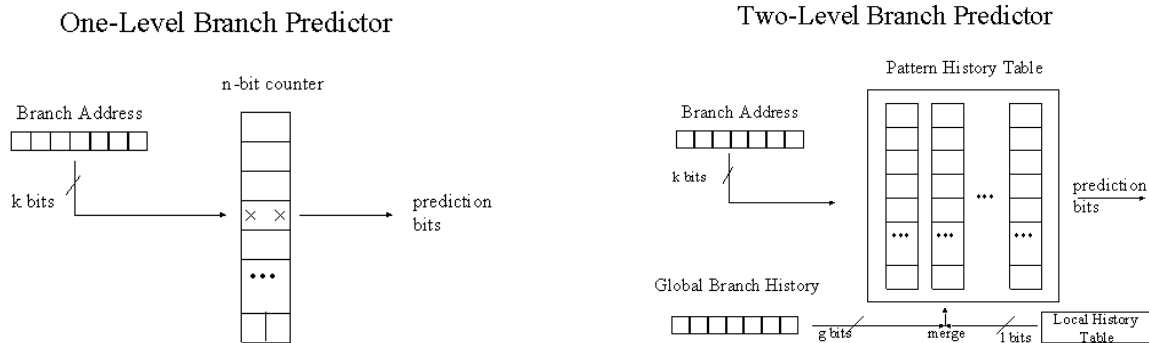
3.3.2 Dynamic Branch Prediction

- The direction of each branch is predicted by recording information, in hardware, of past branch history during a program's execution. ^[7]
- The predictor selects a counter from the table using the lower-order n bits of the instruction's address. The direction prediction is made based on the value of the counter.
- These are typical dynamic prediction scheme :

Decode History Table
Branch History Table
Combination BHT-DHT
Correlation - Based Prediction
Two-Level Adaptive Prediction
Skewed branch predictor
G-share branch predictor

3.4 Branch Prediction & Branch Predictor

In many paper and documents, branch predictions are mixed with predictors. This apparently confused readers. Thus we explain the difference between these two concepts in our paper. As you can see, figures below demonstrate two *dynamic branch predictors* - One-level and Two Level. According to their architecture, you can easily tell these are digit circuits [4].



On the other hand, branch prediction is a strategy in computer architecture design for mitigating the costs usually associated with conditional branches, particularly branches to short sections of code. The following example is a typical logical strategy of branch prediction.

```

if condition                                for (int i=0; t<1000)                if (i>1000)

    do this

else

    do this
  
```


4. Solutions

4.1 Bimodal Predictor Calculation

4.1.1 Bimodal predictor attributes

The bimodal predictor usually serves as the local predictor. It is a state machine with four states:

Strongly not taken
Weakly not taken
Weakly taken
Strongly taken

The bimodal scheme requires the use of a prediction buffer indexed by the program counter value. Makes a prediction based on the direction the branch went the last few times it was executed.

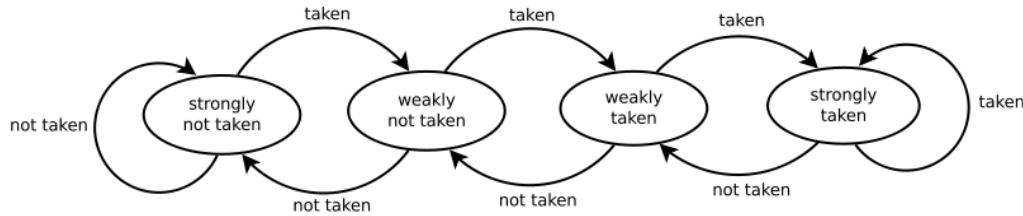


Figure 4.1.1: Bimodal predictor working procedure

4.1.2 Calculate the maximum value of M2

The total memory required for the bimodal predictor $2^{m2} * 2$ bits or $2^{(m2-2)}$ bytes. The total budget is 16 KB i.e. 2^{14} bytes. Equating the limiting conditions $2^{(m2-2)} = 2^{14}$.

The maximum value of M2 = 16.

4.2 G-share Predictor Calculation

In summary, G-share is a variation on the basic global prediction scheme. Global prediction technique uses the combined history of all recent branches in making a prediction. G-share algorithm uses two levels of branch-history information to dynamically predict the direction of the branches. G-share algorithm works by taking the lower bits of the branch target address and

make them work with the history register to get the index that should be used with the prediction table.

4.2.1 Model a G-share branch predictor with parameters $\{m, n\}$

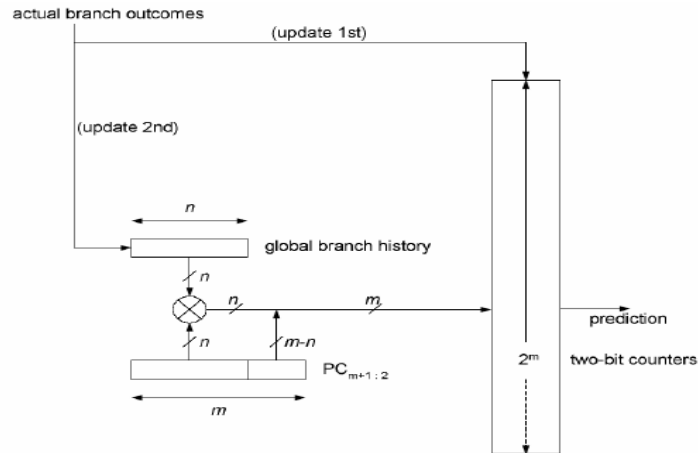


Figure 4.2.1: Building a G-share branch predictor using two parameters

Note: M = The number of low-order PC bits used to form the prediction table index. N = is the number of bits in the global branch history register. $n \leq m$.

4.2.2 Calculate the maximum value of M

The total memory required for the G-share predictor $2^{m1} * 2 \text{ bits} + n \text{ bits}$ or $2^{(m1-2)} + n/8$ bytes. The total budget is 16 KB i.e. 2^{14} bytes. Equating the limiting conditions $2^{(m1-2)} + n/8 = 2^{14}$

$$\because 2^{(m1-2)} \gg n/8, \therefore 2^{(m1-2)} + n/8 \approx 2^{(m1-2)}$$

The maximum value of M = 16.

4.2.3 Two-levels registers

G-shared predictor uses two levels architecture

First Level

- The first level registers the history of the last k branches faced. This level is implemented by providing a global branch history register^[5]
- A shift register that enters a 1 for every taken branch and a 0 for every untaken branch.

Second Level

- The second level of branch history information registers *the branching of the last occurrences of the specific pattern* of the k branches.
- This information is kept in the branch prediction table.

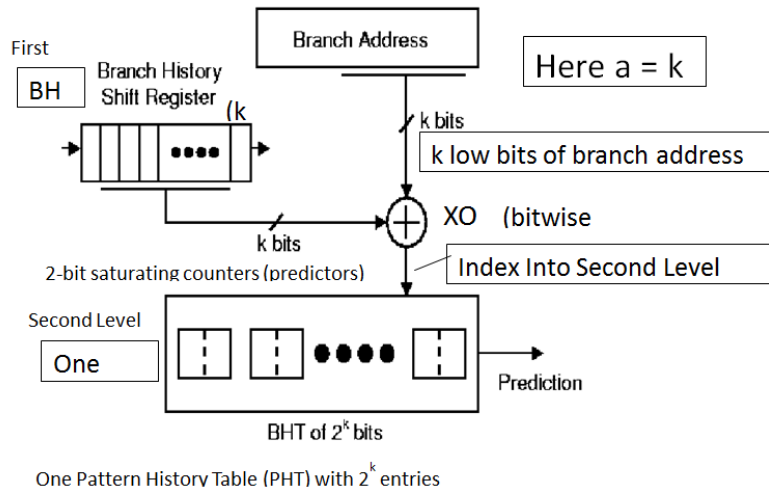


Figure 4.2.3: G-shared Predictor two-level register

4.3 Hybrid Predictor Calculation

4.2.4 Hybrid predictor attributes

Model a hybrid predictor that selects between the bimodal and the G-share predictors, using a chooser table of $2k$ two-bit counters. Here is the selection rule for hybrid predictor.

<i>If the chooser counter = 2 or 3</i>	Selects the G-share
<i>Otherwise</i>	Selects the bimodal

4.2.5 Table update rule and other rules



Result from predictors:	<i>both incorrect or both correct</i>	<i>gshare correct, bimodal incorrect</i>	<i>bimodal correct, gshare incorrect</i>
Chooser counter update policy:	<i>No change</i>	<i>Increment</i>	<i>Decrement</i>

Figure 4.3.2: Update rule

4.2.6 Calculate the maximum value of $M3$

The total memory required for the hybrid predictor is $2^{(m2-2)} + 2^{(m1-2)} + n/8 + 2^{(K-2)}$ bytes.

5. Implementation

5.1 Compiling and Running Simulator

- ❖ The source file designed is compatible to Sun – OS, DOS or Windows
- ❖ Make-file is provided with the package. The source file will automatically get compiled by entering command “make”. The name of the executable simulator file name is “sim”.
- ❖ The simulator simulates the performance of the three different types of the Branch Predictors bimodal, G-share and hybrid. In all the three cases we have to give the command line parameters

5.2 Command line

- *Bio-modal*

```
bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source
$ ./sim bimodal 5 ../resource/gcc_trace.txt
```

Figure 5.2.1: Bio-modal predictor compile command

- sim: simulator program name
- [name] in this case bimodal
- [M2] Branch Program Counter
- [trace file] branch trace text of real program (gcc, jpeg, perl)

▪ ***G-share***

```
bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source  
$ ./sim gshare 5 3 ../resource/gcc_trace.txt
```

Figure 5.2.2: G-share predictor compile command

- sim: simulator program name
- [name] in this case gshare
- [M1] Branch Program Counter
- [N] Global Branch History
- [trace file] branch trace text of real program (gcc, jpeg, perl)

▪ ***Hybrid***

```
bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source  
$ ./sim hybrid 8 5 3 ../resource/gcc_trace.txt
```

Figure 5.2.3: Hybrid predictor compile command

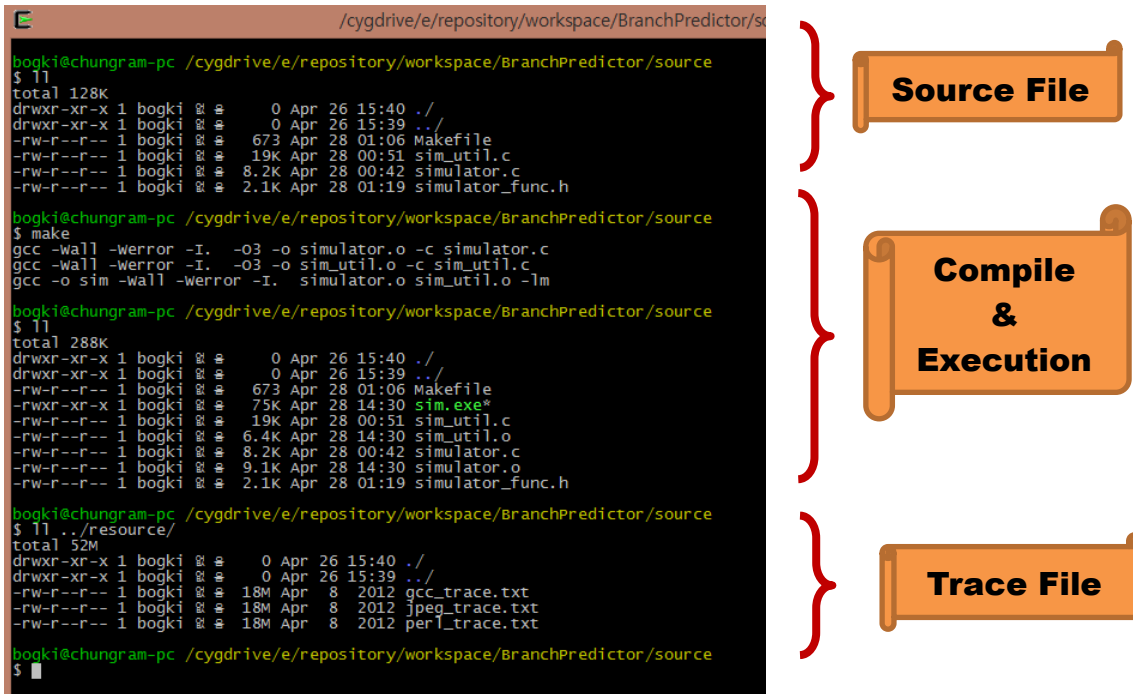
- sim: simulator program name
- [name] in this case hybrid
- [K] Chooser table bits
- [M1] Branch Program Counter is for gshare
- [N] Global Branch History
- [M2] Branch Program Counter is for bimodal
- [trace file] branch trace text of real program (gcc, jpeg, perl)

All these operations are with the following constraints, $0 \leq N \leq M1$.

5.3 Develop Environment

- OS: Windows 8.1
- Compiler: Cygwin for windows GCC
- Editor Tool: Eclipse

5.4 Source and Compiling Example



```

/cygdrive/e/repository/workspace/BranchPredictor/s
bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source
$ ll
total 128K
drwxr-xr-x 1 bogki  0 Apr 26 15:40 ./
drwxr-xr-x 1 bogki  0 Apr 26 15:39 ../
-rw-r--r-- 1 bogki 673 Apr 28 01:06 Makefile
-rw-r--r-- 1 bogki 19K Apr 28 00:51 sim_util.c
-rw-r--r-- 1 bogki 8.2K Apr 28 00:42 simulator.c
-rw-r--r-- 1 bogki 2.1K Apr 28 01:19 simulator_func.h

bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source
$ make
gcc -Wall -Werror -I. -O3 -o simulator.o -c simulator.c
gcc -Wall -Werror -I. -O3 -o sim_util.o -c sim_util.c
gcc -o sim -Wall -Werror -I. simulator.o sim_util.o -lm

bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source
$ ll
total 288K
drwxr-xr-x 1 bogki  0 Apr 26 15:40 ./
drwxr-xr-x 1 bogki  0 Apr 26 15:39 ../
-rw-r--r-- 1 bogki 673 Apr 28 01:06 Makefile
-rwxr-xr-x 1 bogki 75K Apr 28 14:30 sim.exe*
-rw-r--r-- 1 bogki 19K Apr 28 00:51 sim_util.c
-rw-r--r-- 1 bogki 6.4K Apr 28 14:30 sim_util.o
-rw-r--r-- 1 bogki 8.2K Apr 28 00:42 simulator.c
-rw-r--r-- 1 bogki 9.1K Apr 28 14:30 simulator.o
-rw-r--r-- 1 bogki 2.1K Apr 28 01:19 simulator_func.h

bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source
$ ll ./resource/
total 52M
drwxr-xr-x 1 bogki  0 Apr 26 15:40 ./
drwxr-xr-x 1 bogki  0 Apr 26 15:39 ../
-rw-r--r-- 1 bogki 18M Apr 8 2012 gcc_trace.txt
-rw-r--r-- 1 bogki 18M Apr 8 2012 jpeg_trace.txt
-rw-r--r-- 1 bogki 18M Apr 8 2012 perl_trace.txt

bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source
$
  
```

Figure 5.4: screenshot of a compiling example

5.5 Simulator Implement Flowchart

In our implementation, there are four main parts as displayed in the figure below. The first stage initializes variables and many functions and the second stage reading the trace file for comparing real branch and our prediction. The third stage predicts and compares the data generated from real branch and branch predictors. Finally, the last stage displays and outputs the result.

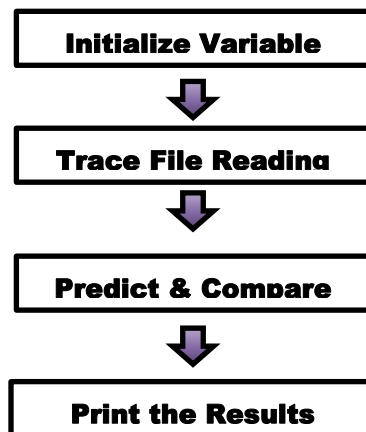


Figure 5.5: Flowchart

5.6 Trace and Trace File Format

We've used Pin¹, a binary instrumentation tool, to generate a trace of branches and their outcomes. We used this representative trace to evaluate the effectiveness of a few simple branch prediction schemes. We extracted trace file from three programs – Gcc, Jpeg, and Perl. In general, most of programs have an unconditional branches and this is always taken, so that we excluded it from this trace file. ^[6]

Spec95 Executable	Processor Focus	Description	Trace File Name
Gcc	Integer & Branch Intensive	A C software language compiler. Arguably one of the most used software packages worldwide.	gcc_trace.txt
Jpeg	Integer & Branch Intensive	Image compression and decompression	jpeg_trace.txt
Perl	Integer & Branch Intensive	Programing Language	perl_trace.txt

The simulator reads a trace file in the following format:

```
0x<hex branch PC> t/n
0x<hex branch PC> t/n
```

.. where <hex branch PC> holds the hexadecimal form of address of the branch instruction in memory.

“t” indicates that the branch is actually taken.

“n” indicates that the branch is not taken.

Example: 1100023b; 0a050201n

¹ Pin is a kind of program Instrumentation and Profiling(<https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>)

5.7 Results Format

The simulator outputs the following statistics after completion of the run: 1.Total Number of
2. Branches Number of Mis-predictions 3.Mis-prediction Rate (Number of Mis-predictions /
Total number of branches) 4. Final Counter Contents

```
/cygdrive/e/repository/workspace/BranchPredictor

bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source
$ ./sim bimodal 5 ../resource/gcc_trace.txt
COMMAND
./sim bimodal 5 ../resource/gcc_trace.txt
OUTPUT
number of predictions: 2000000
number of mispredictions: 685835
misprediction rate: 34.29%
FINAL BIMODAL CONTENTS
0      3
1      1
2      3
3      0
4      0
5      2
6      0
7      0
8      2
9      2
10     0
11     2
12     2
13     0
14     1
15     3
16     1
17     1
18     3
19     2
20     0
21     3
22     1
23     3
24     0
25     2
26     0
27     1
28     0
29     1
30     0
31     0

bogki@chungram-pc /cygdrive/e/repository/workspace/BranchPredictor/source
$
```

Figure 5.7: Result format of output

6. Evaluation and Results

6.1 The Parameters in the Cache Simulation

The cache performance results were calculated with the three trace files: gcc_trace.txt, jpeg_trace.txt, perl_trace.txt.

File Name	Bimodal Predictor	Gshare Predictor	
	M2 ($7 \leq M2 \leq 16$)	M1 ($7 \leq M1 \leq 16$)	$2 \leq N \leq M1$
gcc_trace.txt	7	7	2
jpeg_trace.txt	8	8	3
perl_trace.txt	9	9	4
	10	10	5
	11	11	6
	12	12	7
	13	13	8
	14	14	9
	15	15	10
	16	16	11
			12
			13
			14
			15
			16

Figure 6.1: Parameters of the cache simulation

The predictor parameters M2 (in case of bimodal), M1 and N (in case of G-share) are varied in the simulation. The following values are considered for the simulation. Since the allocated budget for the predictor was 16KB, the maximum “M1” value that can be chosen is 16.

6.2 Trace1: gcc_trace.txt

Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate (R1)
bimodal	7	-	-	2000000	532970	26.65
	8	-	-	2000000	448572	22.43
	9	-	-	2000000	369866	18.49
	10	-	-	2000000	313427	15.67
	11	-	-	2000000	273003	13.65
	12	-	-	2000000	249344	12.47
	13	-	-	2000000	234386	11.72
	14	-	-	2000000	227415	11.37
	15	-	-	2000000	225919	11.30
	16	-	-	2000000	224123	11.21

Figure 6.2: Misprediction Rate of Bimodal Predictor statistical table for gcc_trace.txt

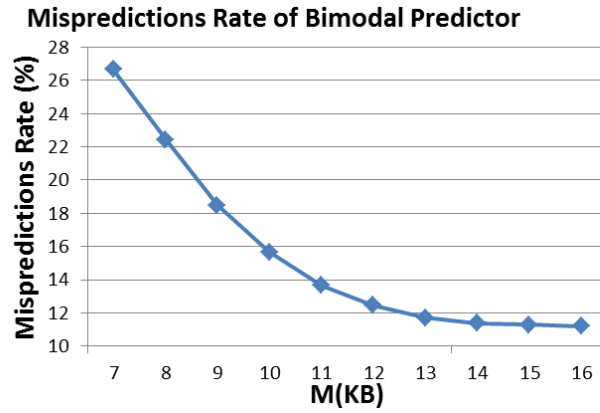


Figure 6.3: Misprediction Rate of Bimodal Predictor chart for gcc_trace.txt

Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate (R1)
gshare	-	7	2	2000000	579675	28.98
	-	8	2	2000000	503567	25.18
	-	9	2	2000000	404948	20.25
	-	10	2	2000000	327851	16.39
	-	11	2	2000000	274296	13.71
	-	12	2	2000000	243917	12.2
	-	7	4	2000000	615177	30.76
	-	8	4	2000000	531458	26.57
	-	9	4	2000000	448594	22.43
	-	10	4	2000000	359849	17.99
	-	11	4	2000000	289768	14.49
	-	12	4	2000000	244675	12.23
	-	7	6	2000000	664406	33.22
	-	8	6	2000000	556476	27.82
	-	9	6	2000000	482866	24.14
	-	10	6	2000000	387262	19.36
	-	11	6	2000000	302761	15.14
	-	12	6	2000000	249123	12.46
	-	8	8	2000000	611232	30.56
	-	9	8	2000000	521585	26.08
	-	10	8	2000000	422021	21.1
	-	11	8	2000000	329365	16.47
	-	12	8	2000000	260043	13
	-	10	10	2000000	455390	22.77
	-	11	10	2000000	366769	18.34
	-	12	10	2000000	286523	14.33
	-	12	12	2000000	308098	15.4

Figure 6.4: Misprediction Rate of Gshare Predictor statistical table for gcc_trace.txt



Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate (R1)
gshare $13 \leq M1 \leq 16$	-	13	2	2000000	222102	11.11
	-	14	2	2000000	208474	10.42
	-	15	2	2000000	202348	10.13
	-	16	2	2000000	198515	9.93
	-	13	4	2000000	211351	10.57
	-	14	4	2000000	193749	9.69
	-	15	4	2000000	182655	9.13
	-	16	4	2000000	175470	8.77
	-	13	6	2000000	211873	10.59
	-	14	6	2000000	181578	9.08
	-	15	6	2000000	166079	8.3
	-	16	6	2000000	157787	7.89
	-	13	8	2000000	219904	11
	-	14	8	2000000	186773	9.34
	-	15	8	2000000	164321	8.22
	-	16	8	2000000	151306	7.57
	-	13	10	2000000	233535	11.68
	-	14	10	2000000	196660	9.83
	-	15	10	2000000	169199	8.46
	-	16	10	2000000	152164	7.61
	-	13	12	2000000	253554	12.68
	-	14	12	2000000	209513	10.48
	-	15	12	2000000	180288	9.01
	-	16	12	2000000	157281	7.86
	-	14	14	2000000	222673	11.13
	-	15	14	2000000	189624	9.48
	-	16	14	2000000	166826	8.34
	-	16	16	2000000	174928	8.75

Figure 6.5: Misprediction Rate of Gshare Predictor statistical table for gcc_trace.txt

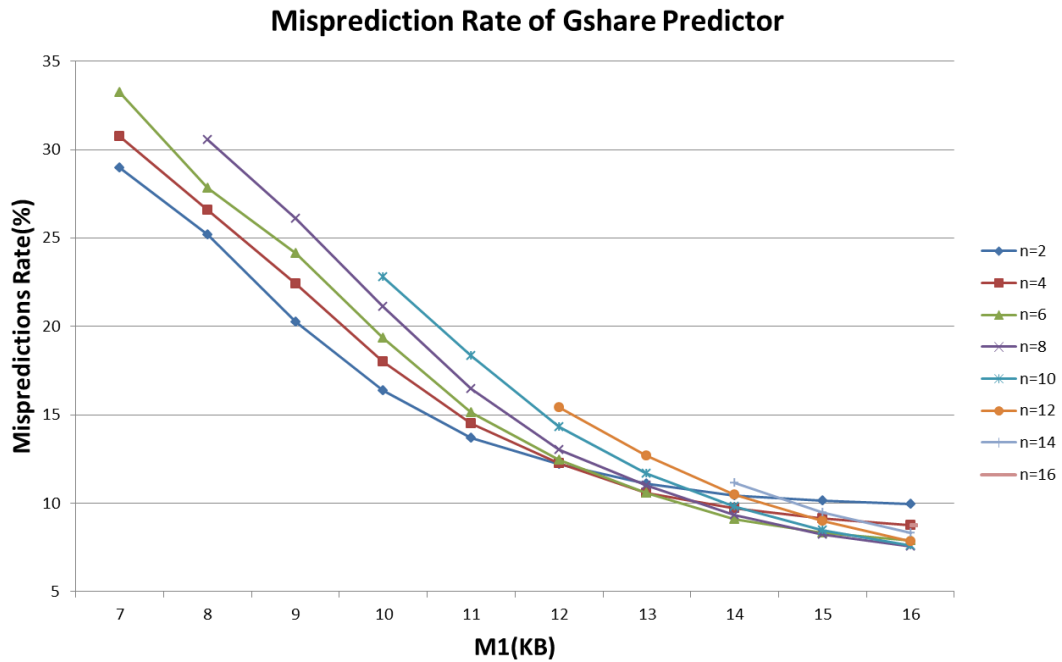


Figure 6.6: Misprediction Rate of Gshare Predictor chart for gcc_trace.txt

6.3 Trace2: jpeg_trace.txt

Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate (R1)
bimodal	7	-	-	2000000	158454	7.92
	8	-	-	2000000	155757	7.79
	9	-	-	2000000	154753	7.74
	10	-	-	2000000	154097	7.7
	11	-	-	2000000	152314	7.62
	12	-	-	2000000	151950	7.6
	13	-	-	2000000	151818	7.59
	14	-	-	2000000	151753	7.59
	15	-	-	2000000	151704	7.59
	16	-	-	2000000	151705	7.59

Figure 6.7: Misprediction Rate of Bimodal Predictor statistical table for jpeg_trace.txt

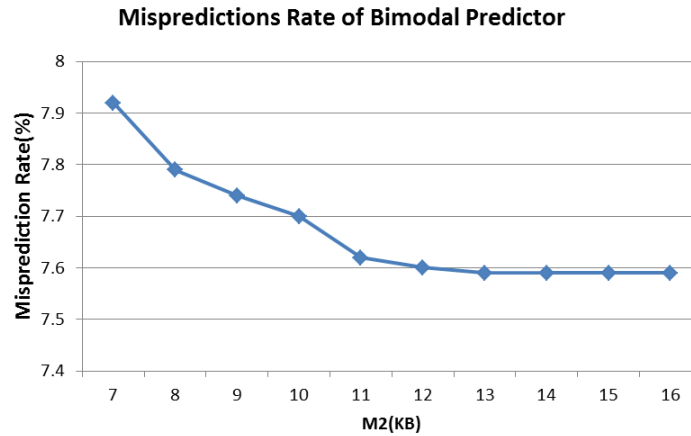


Figure 6.8: Misprediction Rate of Bimodal Predictor chart for jpeg_trace.txt

Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate (R1)
gshare	-	7	2	2000000	161581	8.08
	-	8	2	2000000	178396	8.92
	-	9	2	2000000	194735	9.74
	-	10	2	2000000	155708	7.79
	-	11	2	2000000	157505	7.88
	-	12	2	2000000	177337	8.87
	-	7	4	2000000	184042	9.2
	-	8	4	2000000	151619	7.58
	-	9	4	2000000	153519	7.68
	-	10	4	2000000	162637	8.13
	-	11	4	2000000	166058	8.3
	-	12	4	2000000	149728	7.49
	-	7	6	2000000	147514	7.38
	-	8	6	2000000	151513	7.58
	-	9	6	2000000	149061	7.45
	-	10	6	2000000	158951	7.95
	-	11	6	2000000	148947	7.45
	-	12	6	2000000	145362	7.27
	-	8	8	2000000	147639	7.38
	-	9	8	2000000	143348	7.17
	-	10	8	2000000	148866	7.44
	-	11	8	2000000	148835	7.44
	-	12	8	2000000	145193	7.26
	-	10	10	2000000	143752	7.19
	-	11	10	2000000	136893	6.84
	-	12	10	2000000	143647	7.18
	-	12	12	2000000	147072	7.35

Figure 6.9: Misprediction Rate of Gshare Predictor statistical table for jpeg_trace.txt



Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate (R1)
gshare $13 \leq M1 \leq 16$	-	13	2	2000000	146570	7.33
	-	14	2	2000000	144892	7.24
	-	15	2	2000000	143236	7.16
	-	16	2	2000000	136592	6.83
	-	13	4	2000000	140362	7.02
	-	14	4	2000000	143374	7.17
	-	15	4	2000000	146428	7.32
	-	16	4	2000000	143314	7.17
	-	13	6	2000000	142751	7.14
	-	14	6	2000000	133797	6.69
	-	15	6	2000000	136835	6.84
	-	16	6	2000000	136727	6.84
	-	13	8	2000000	138532	6.93
	-	14	8	2000000	146296	7.31
	-	15	8	2000000	142610	7.13
	-	16	8	2000000	141713	7.09
	-	13	10	2000000	133733	6.69
	-	14	10	2000000	134375	6.72
	-	15	10	2000000	134053	6.7
	-	16	10	2000000	133377	6.67
	-	13	12	2000000	146226	7.31
	-	14	12	2000000	142524	7.13
	-	15	12	2000000	141641	7.08
	-	16	12	2000000	133008	6.65
	-	14	14	2000000	133978	6.7
	-	15	14	2000000	133235	6.66
	-	16	14	2000000	131461	6.57
	-	16	16	2000000	133686	6.68

Figure 6.10: Misprediction Rate of Gshare Predictor statistical table for jpeg_trace.txt

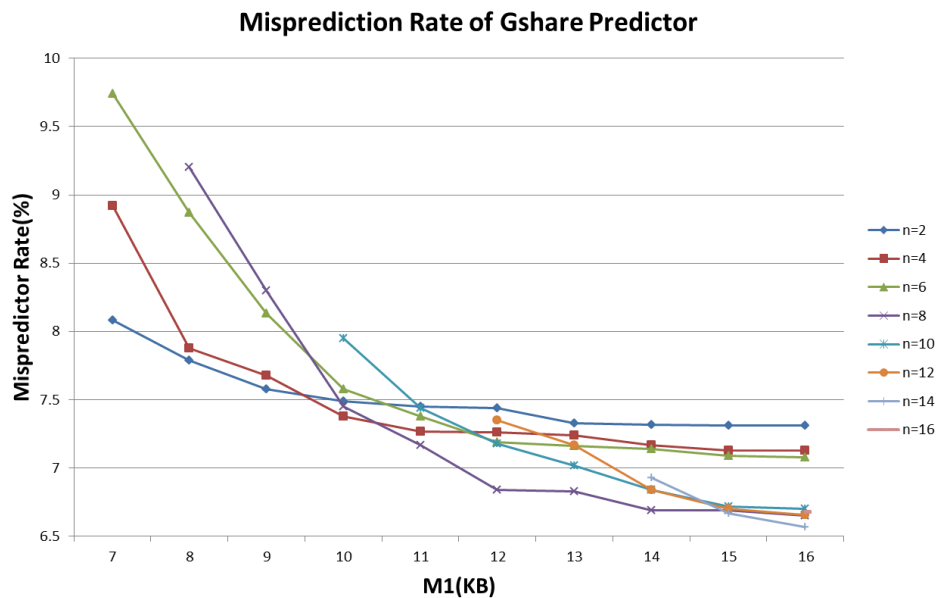


Figure 6.11: Misprediction Rate of Gshare Predictor chart for jpeg_trace.txt

6.4 Trace3: perl_trace.txt

Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate(R1)
bimodal	7	-	-	2000000	426289	21.31
	8	-	-	2000000	329056	16.45
	9	-	-	2000000	282848	14.14
	10	-	-	2000000	238913	11.95
	11	-	-	2000000	220928	11.05
	12	-	-	2000000	181800	9.09
	13	-	-	2000000	178484	8.92
	14	-	-	2000000	176448	8.82
	15	-	-	2000000	176386	8.82
	16	-	-	2000000	176621	8.83

Figure 6.12: Misprediction Rate of Bimodal Predictor statistical table for perl_trace.txt

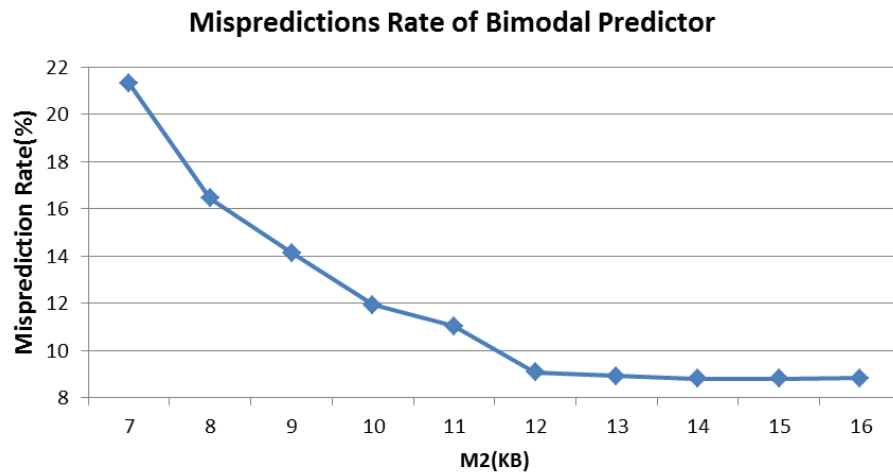


Figure 6.13: Mis-prediction Rate of Bimodal Predictor chart for perl_trace.txt

Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate(R1)
gshare	-	7	2	2000000	486879	24.34
	-	8	2	2000000	338433	16.92
	-	9	2	2000000	271429	13.57
	-	10	2	2000000	212689	10.63
	-	11	2	2000000	202126	10.11
	-	12	2	2000000	180545	9.03
	-	7	4	2000000	519197	25.96
	-	8	4	2000000	381794	19.09
	-	9	4	2000000	293687	14.68
	-	10	4	2000000	226958	11.35
	-	11	4	2000000	193603	9.68
	-	12	4	2000000	161749	8.09
	-	7	6	2000000	574206	28.71
	-	8	6	2000000	409087	20.45
	-	9	6	2000000	324956	16.25
	-	10	6	2000000	230495	11.52
	-	11	6	2000000	172036	8.6
	-	12	6	2000000	149959	7.5
	-	8	8	2000000	495760	24.79
	-	9	8	2000000	353184	17.66
	-	10	8	2000000	248331	12.42
	-	11	8	2000000	180021	9
	-	12	8	2000000	129841	6.49
	-	10	10	2000000	291319	14.57
	-	11	10	2000000	179598	8.98
	-	12	10	2000000	134202	6.71
	-	12	12	2000000	143222	7.16

Figure 6.14: Misprediction Rate of Gshare Predictor statistical table for perl_trace.txt

Predictor Type	M2	M1	N	Total Branches	Total Mispredictions	Mispredictions Rate(R1)
gshare $13 \leq M1 \leq 16$	-	13	2	2000000	184575	9.23
	-	14	2	2000000	145331	7.27
	-	15	2	2000000	121723	6.09
	-	16	2	2000000	105279	5.26
	-	13	4	2000000	98492	4.92
	-	14	4	2000000	101816	5.09
	-	15	4	2000000	161408	8.07
	-	16	4	2000000	147094	7.35
	-	13	6	2000000	108687	5.43
	-	14	6	2000000	90198	4.51
	-	15	6	2000000	73966	3.8
	-	16	6	2000000	86034	4.3
	-	13	8	2000000	75040	3.75
	-	14	8	2000000	160408	8.02
	-	15	8	2000000	145603	7.28
	-	16	8	2000000	114204	5.71
	-	13	10	2000000	82676	4.13
	-	14	10	2000000	71675	3.58
	-	15	10	2000000	67000	3.35
	-	16	10	2000000	71652	3.58
	-	13	12	2000000	160769	8.04
	-	14	12	2000000	130835	6.54
	-	15	12	2000000	101371	5.07
	-	16	12	2000000	82363	4.12
	-	14	14	2000000	76838	3.84
	-	15	14	2000000	70589	3.53
	-	16	14	2000000	60148	3.01
	-	16	16	2000000	58189	2.91

Figure 6.15: Misprediction Rate of Gshare Predictor statistical table for perl_trace.txt

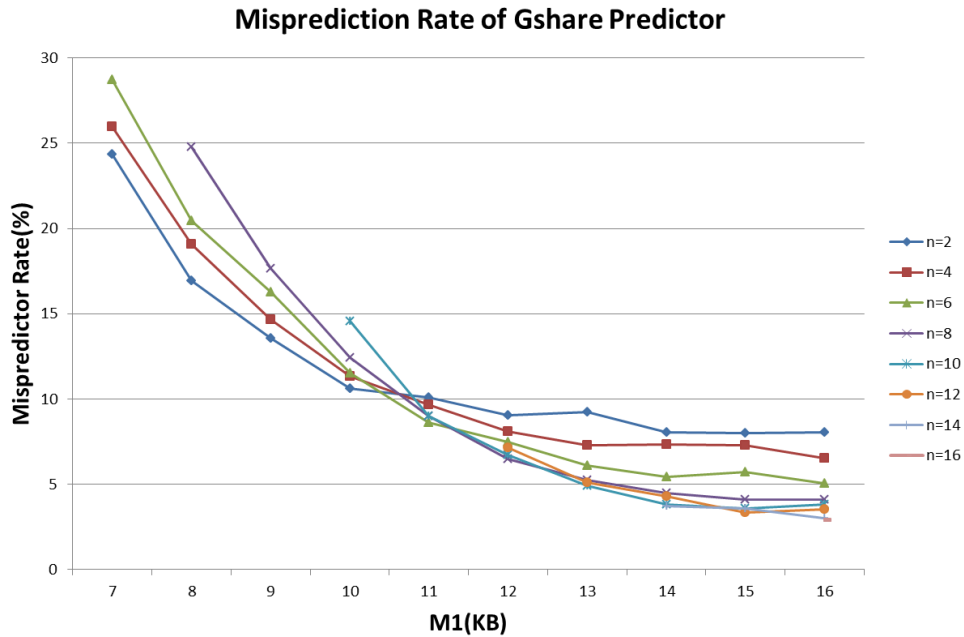


Figure 6.16: Misprediction Rate of Gshare Predictor chart for perl_trace.txt

6.5 Combined results for bimodal predictor

Predictor Type	M2	Total Branches	Total Mispredictions	Mispredictions Rate (R1)
gcc	7	2000000	532970	26.65
gcc	8	2000000	448572	22.43
gcc	9	2000000	369866	18.49
gcc	10	2000000	313427	15.67
gcc	11	2000000	273003	13.65
gcc	12	2000000	249344	12.47
gcc	13	2000000	234386	11.72
gcc	14	2000000	227415	11.37
gcc	15	2000000	225919	11.3
gcc	16	2000000	224123	11.21
jpeg	7	2000000	158454	7.92
jpeg	8	2000000	155757	7.79
jpeg	9	2000000	154753	7.74
jpeg	10	2000000	154097	7.7
jpeg	11	2000000	152314	7.62
jpeg	12	2000000	151950	7.6
jpeg	13	2000000	151818	7.59
jpeg	14	2000000	151753	7.59
jpeg	15	2000000	151704	7.59
jpeg	16	2000000	151705	7.59
perl	7	2000000	426289	21.31
perl	8	2000000	329056	16.45
perl	9	2000000	282848	14.14
perl	10	2000000	238913	11.95
perl	11	2000000	220928	11.05
perl	12	2000000	181800	9.09
perl	13	2000000	178484	8.92
perl	14	2000000	176448	8.82
perl	15	2000000	176386	8.82
perl	16	2000000	176621	8.83

Figure 6.17: Mis-prediction Rate of Bimodal Predictor statistical table

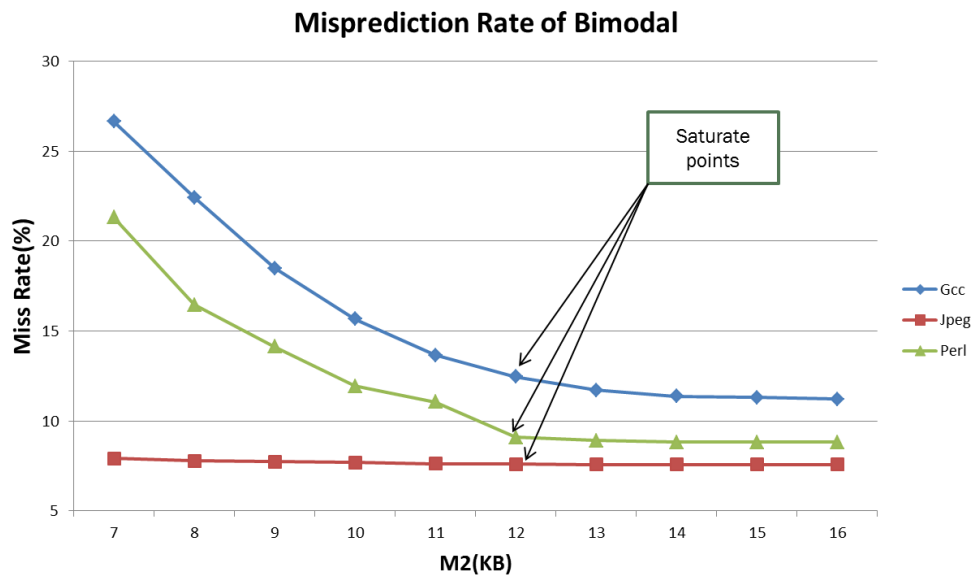


Figure 6.18: Misprediction Rate of Bimodal Predictor chart

As it can be seen from the above graph, the mis-prediction rate decreases as we increase m2. But the reduction rates and the rate at which the mis-prediction rate is decreased are different for gcc, jpeg and perl traces. So we can conclude that the mis-prediction rate is highly dependent on the nature of the code. The reduction in mis-prediction rate is the least for jpeg trace. The gcc and perl traces show significant reduction in mis-prediction rate when m2 is increased. One possible reason for this is jpeg trace contains lesser branches compared to gcc and perl so the reduction in jpeg trace is negligible or the predictor. We can see from the graph that there is not much reduction in the rate after m2 = 12. So this point of the graph can be considered as the saturation point for the bimodal predictor. So I choose the design of bimodal predictor with m2 = 12.

6.6 Combined results for G-share predictor

Predict or Type	N	M1	Total Branches	Total Mispredictions	Mispredictions Rate(R1)
gcc	16	2	2000000	198515	9.93
	16	4	2000000	175470	8.77
	16	6	2000000	157787	7.89
	16	8	2000000	151306	7.57
	16	10	2000000	152164	7.61
	16	12	2000000	157281	7.86
	16	14	2000000	166826	8.34
	16	16	2000000	174928	8.75
jpeg	16	2	2000000	146226	7.31
	16	4	2000000	142524	7.13
	16	6	2000000	141641	7.08
	16	8	2000000	133008	6.65
	16	10	2000000	133978	6.7
	16	12	2000000	133235	6.66
	16	14	2000000	131461	6.57
	16	16	2000000	133686	6.68
perl	16	2	2000000	160769	8.04
	16	4	2000000	130835	6.54
	16	6	2000000	101371	5.07
	16	8	2000000	82363	4.12
	16	10	2000000	76838	3.84
	16	12	2000000	70589	3.53
	16	14	2000000	60148	3.01
	16	16	2000000	58189	2.91

Figure 6.19: Misprediction Rate of Gshare Predictor chart

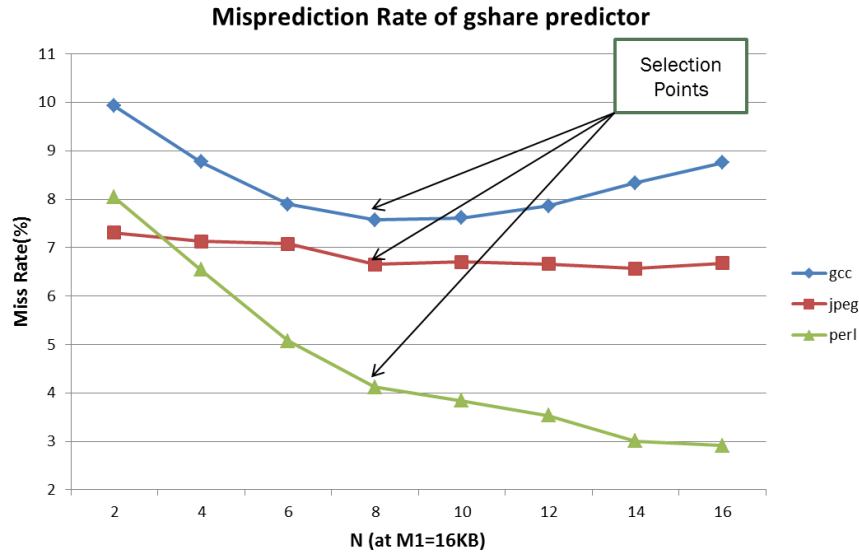


Figure 6.20: Misprediction Rate of Gshare Predictor chart

Similar inference like to bimodal can be drawn here the misprediction rate decreases as we increase m1 and n. It is seen from the comparison of the graphs of bimodal and gshare that gshare predictor has higher misprediction rate compared to bimodal predictor for low values of m. but ultimately as we go on increasing the value of m2 and n. The reduction in misprediction rate is more prominent in gshare predictor compared to bimodal predictor, i.e. for the given memory size, gshare predictor has lower misprediction rates than bimodal. We can see from the graph that in all the three traces, the least reduction rate is achieved at $m1 = 16$. Graph represents the variation of misprediction rate against n for $m = 16$. It is observed that at $n = 8$, both gcc and jpeg approximately reach to their lowest point and perl continues to decreased for all n. So looking at the trend $n = 8$ is the point of the graph that can be considered as the selection point for the gshare predictor.

The Misprediction Rate recorded at M1=16, N=8				The Lowest Record of Misprediction Rate			
	Trace Name				Trace Name		
	GCC	JPEG	PERL		GCC	JPEG	PERL
Mispredictions Rate(%)	7.57	6.65	4.12	M1	16	16	16
				N	8	14	16
				Mispredictions Rate(%)	7.57	6.57	2.91

Figure 6.21: Misprediction Rate of Gshare Predictor record table

Overall selection for bimodal predictor

M2-12 bits (Total memory - $2^{12} * 2$ bits – 1KB)

Overall selection for G-share predictor

M1-16 bits (Total memory - $2^{16} * 2$ bits – 16KB+8 bits)



7. Conclusions

The mis-prediction rate of a branch predictor decrease as the values of the branch predictor's parameters increased. It also increases the predictor's size. The performance tends to saturate after a certain point and then the improvement becomes relatively small for larger parameters. The simulation demonstrates that benchmarks defer in their behavior resulting different mis-prediction rates. The types of applications used largely affect a branch predictor's behavior.



Reference

- [1] Smith, J.E., “A Study of Branch Prediction Strategies”, *Proceedings of the 8th annual symposium on Computer Architectures 1981*, IEEE Computer Society Press: Minneapolis, Minnesota, USA. p. 135-148
- [2] Yeh, T.-Y. and Y.N. Patt, “Two-level adaptive training branch prediction”, *Proceedings of the 24th annual international symposium on Microarchitecture* 1991.
- [3] McFarling, “Combining Branch Predictors”, *WRL Technical Note TN-36*, Digital Equipment Corporation, June 1993
- [4] Michaud, Pierre, Andre Siznec, Richard Uhlig, “Skewed branch predictors”, *Tech. report, IRISA publ. int. 1031*, June 1996
- [5] Michaud, P., et al., “Trading conflict and capacity Aliasing in conditional branch predictors”, *Proceedings of the 24th annual international symposium on Computer architecture*, 1997.
- [6] McFarling, S., “Combining branch predictors”, *Compaq Computer Corporation Western Research Laboratory*, 1993.
- [7] D. Burger, T.M.A., and S.Bennett, “Evaluating future micro-processors: the SimpleSclar tool set”