VIKRAM SRI NITESH TANTRAVAHI
ID: 13247361

Simulation Project Part 1

Introduction to GEM5 and Branch Prediction

Introduction:

There is a lot of complexity that exists in the Instruction Set Architecture (ISA) because of intense and deep pipelining and branching techniques that are implemented that eventually ends up in stalls. So the Branch prediction helps us eliminate this problem as it tries to guess which way a branch goes before its way is known. It plays a very important and crucial role in achieving high effective performance in most of the advanced pipelined microprocessor architectures. This project is simulated and aims at analyzing various branch prediction schemes against different workloads. It also allows us to understand how branch prediction affects CPU performance and how is it evaluated.

Experimental Setup:

This experiment uses GEM5 to investigate dynamic branch predictors namely, bimodal and tournament. The experiment is conducted with three different predictors *Taken*, *Bimodal* and *Tournament* along with two benchmarks, Perlbench and GCC that are configured by specifying type to *local* or *T*, setting local and global predictor sizes and counter bits. The statistics and processor configuration are saved in config.ini, config.json and stats.txt for each benchmark. The statistics and results that are reported are IPC, execution cycles, branch rate, branch misprediction and percentage fetch-squashed etc.

Results and analysis:

1. **CPU configuration:**

   The table below depicts the CPU configuration along with the obtained results.
   The desired characteristics that are recorded are calculated using specific terms from *stats.txt*.
   They are:

   a. IPC (system.cpu.ipc)
   b. Execution cycles (system.cpu.numCycles)
   c. Branch rate (system.cpu.fetch.branchRate)
   d. Branch mis-prediction rate (system.cpu.branchPred.condIncorrect) / (system.cpu.branchPred.condPredicted)
   e. % fetch-squashed (system.cpu.fetch.SquashCycles) / (system.cpu.fetch.Cycles)

	Predict Taken		**N-bit local (bimodal) branch predictor**						**Tournament predictor**	
	2048_2bits		**1024_1bit**		**1024_2bits**		**65536_2bits**			
Benchmark	Perl bench	GCC	Perl bench	GCC	Perl bench	GCC	Perl bench	GCC	Perl bench	GCC
cache_line_size	64	64	64	64	64	64	64	64	64	64
load_addr_mask	1099511627775	1099511627775	1099511627775	1099511627775	1099511627775	1099511627775	1099511627775	1099511627775	1099511627775	1099511627775
mem_ranges	0:2147483647	0:2147483647	0:2147483647	0:2147483647	0:2147483647	0:2147483647	0:2147483647	0:2147483647	0:2147483647	0:2147483647
num_work_ids	16	16	16	16	16	16	16	16	16	16
clock	333	333	333	333	333	333	333	333	333	333
Fetch Buffer Size	64	64	64	64	64	64	64	64	64	64

VIKRAM SRI NITESH TANTRAVAHI
ID: 13247361

type	DerivO3 CPU	DerivO3 CPU	DerivO3 CPU	DerivO3 CPU	DerivO3C PU	DerivO3 CPU	DerivO3 CPU	DerivO3 CPU	DerivO3 CPU	DerivO3 CPU
smtROBPolicy	Partioned	Partioned	Partioned	Partioned	Partioned	Partioned	Partioned	Partioned	Partioned	Partioned
localCtrBits	2	2	1	1	2	2	2	2	3	3
LocalHistoryTableSize	2048	2048	2048	2048	2048	2048	2048	2048	1024	1024
localPredictorSize	2048	2048	1024	1024	1024	1024	65536	65536	1024	1024
numThreads	1	1	1	1	1	1	1	1	1	1
predType	T	T	local	Local	local	Local	Local	Local	Tournament	Tournament
IPC	0.483141	0.450264	0.544542	0.461711	0.539864	0.461697	0.564650	0.461713	0.558193	0.458170
Execution cycles	4357182	555230170	3865883	541464803	3901537	541480529	3730276	541462396	3773425	545648460
Branch rate	0.211846	0.227735	0.216964	0.075089	0.215175	0.075014	0.215390	0.075018	0.207691	0.077418
Branch mispredictio n rate	124864/ 923050 0.135273	16233573/ 12644509 0.1283843	86309 / 838248 0.10293	98769 / 40658266 0.00242	88576/ 839514 0.1055	92334/ 40618625 0.02273	75624/ 803464 0.094122	92479/ 40619505 0.00227	80135/ 783708 0.1022	367943/ 42242881 0.00871
% fetch squashed	488510/ 1333622 0.366303	78035732/ 18605696 0.4194185	371263 / 1142083 0.32507	775876 / 66254550 0.00117	376575 / 1145158 0.3288	613278/ 66084622 0.00928	332311/ 1085502 0.3061	615138 / 66089134 0.0093077	334649 / 1075491 0.3111	2214417/ 68829512 0.03217

Table 1: CPU Configuration

2. Workloads of Spec CPU 2006.

Perlbench:
400.Perlbench is a refined version of a popular scripting language. This workload consists of three scripts:
1. Spam Assasin, which is open source software.
2. MHonArc, which is an email-to HTML converter.
3. A script that has 'specdiff' script which uses mail generator. The output can be classified based on the cases. In case of mail-based benchmarks, a line describing the properties of output message is generated. In case of processing, MD5 contents are computed.
In SpamAssasin, the message score and the rules triggered are described.

GCC:
It consists of 9 workloads. They are the preprocessed C code files.
They are cp-decl.i, expr.i,166.i, 200.i, scilab.i, Expr2.i, c-typeck.i, g23.i, and s04.

3. Predictor size in terms of storage:

The predictor has local predictor size of 1024 bits with counter size 1 bit and 2 bits. The configuration of predictor files are also changed by changing global predictor size from 8192 bits to 4096 bits with the counter bits of 1,2 or 3 bits.

Predict Taken "predictor":T:

LocalPredictorSize : 2048
LocalCtrBits : 2
LocalHistoryTableBits :N/A
globalPredictorSize :N/A
globalCtrBits :N/A

VIKRAM SRI NITESH TANTRAVAHI
ID: 13247361

N bit local (bimodal) branch predictor:

1 bit:
LocalPredictorSize :1024
LocalCtrBits :1
LocalHistoryTableBits : N/A
globalPredictorSize :N/A
globalCtrBits :N/A

2 bit:
LocalPredictorSize :1024
LocalCtrBits :2
LocalHistoryTableBits : N/A
globalPredictorSize :N/A
globalCtrBits :N/A

2 bit:
LocalPredictorSize :65536
LocalCtrBits :2
LocalHistoryTableBits : N/A
globalPredictorSize :N/A
globalCtrBits :N/A

Tournament:

LocalPredictorSize :1024
LocalCtrBits :2
LocalHistoryTableBits : 1024
globalPredictorSize :4096
globalCtrBits :2

4. Working of Predictor

Static prediction is the simplest br anch prediction technique as it depends solely on branch instruction. used single direction static branch prediction: they always predicted that a conditional jump would not be taken, so they always fetched the next sequential instruction. Only when the branch or jump was evaluated and found to be taken did the instruction pointer get set to a non-sequential address. This is illustrated in fig 1.

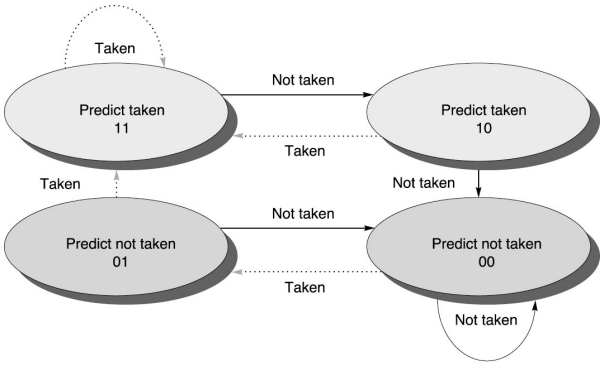


Fig. 1

The Bimodal counter has four states, strongly not taken, strongly taken, weakly not taken and weakly taken. Statemachine is updatedwhen a branch is evaluated. The advantage of 2 bit counter over 1 bit. When a

conditional branch occurs, a two-bit counter assigns a two bit counter to prediction buffer. When the branch is taken the counter is incremented else decremented. A prediction must be wrong twice before it is changed. The prediction is determined by the most significant bit. History table is maintained to collect information and uses this information to make a prediction. The state of branch's entry in the buffer is therefore changed dynamically when the branch instructions are executed. Two bit prediction scheme is illustrated in the figure 2 below.

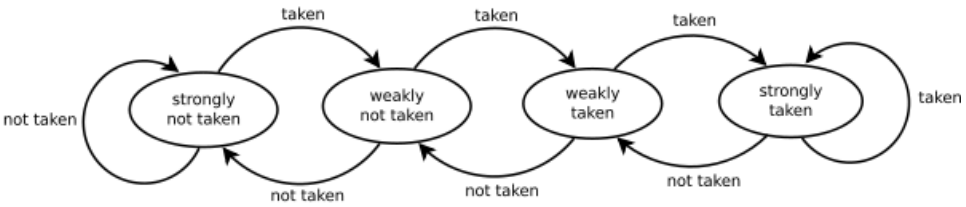


Fig. 2

5. Predictor implemented in Gem5

The BranchPrediction.py code is executed this way. First various classes are called for which the configuration parameters are passed. Two kinds of classes local and global are used. The following code explains how the Taken predictor predicts the branch to be taken.

```
from m5.SimObject import SimObject
from m5.params import *

class BranchPredictor(SimObject):
// "SimObject" class declaration
 type = 'BranchPredictor'
 cxx_class = 'BPredUnit'
// " BPredUnit" class declaration
 cxx_header = "cpu/pred/bpred_unit.hh"
 // header declaration
 numThreads = Param.Unsigned(1, "Number of threads")
//passing value 1 to numthread function
 predType = Param.String("T", // predictor selection
"Branch predictor type ('local', 'tournament','T','NT','ECEC621')")
// various predictors listed
localPredictorSize = Param.Unsigned(2048, "Size of local predictor")
// calling and passing local variable 2048 to it
 localCtrBits = Param.Unsigned(2, "Bits per counter")
 //'2' control bits are passed to 'localctrbits', a local variable
 localHistoryTableSize = Param.Unsigned(1024, "Size of local history table")
// passing '1024'bits to "localhistorytablesize"
 globalPredictorSize = Param.Unsigned(4096, "Size of global predictor")
// passing 4096 bits to 'globalpredictorsize'.Its a local variable.
 globalCtrBits = Param.Unsigned(2, "Bits per counter")
// passing 2 bit control bits to global variable to globalctrbits
choicePredictorSize = Param.Unsigned(8192, "Size of choice predictor")
// passing '8192' bits to 'choicepredictorsize'
choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")
// passing '2' bits to 'choicectrbits'-counters
BTBEntries = Param.Unsigned(4096, "Number of BTB entries")
// passing '2' bits to 'BTBEntries'
BTBTagSize = Param.Unsigned(16, "Size of the BTB tags, in bits")
// calling the RASSize class passing the value of 16
RASSize = Param.Unsigned(16, "RAS size")
```

VIKRAM SRI NITESH TANTRAVAHI
ID: 13247361

//calling the intShiftAmt class passing the value of 2 .

This explains the amount of bits by which the data has to be shifted.

6. Changing counter size affects predictors accuracy

From predictor size, we can find the number of prediction bits. By increasing the predictor size, we can say that the number of predictor bits has been increased, which affects the branch prediction accuracy. Increasing the number of prediction bits does not necessarily improve the branch prediction accuracy; since it also depends on the benchmark, cache sizes, etc. It is possible that the prediction accuracy continues to increase as the number of prediction bits is increased.

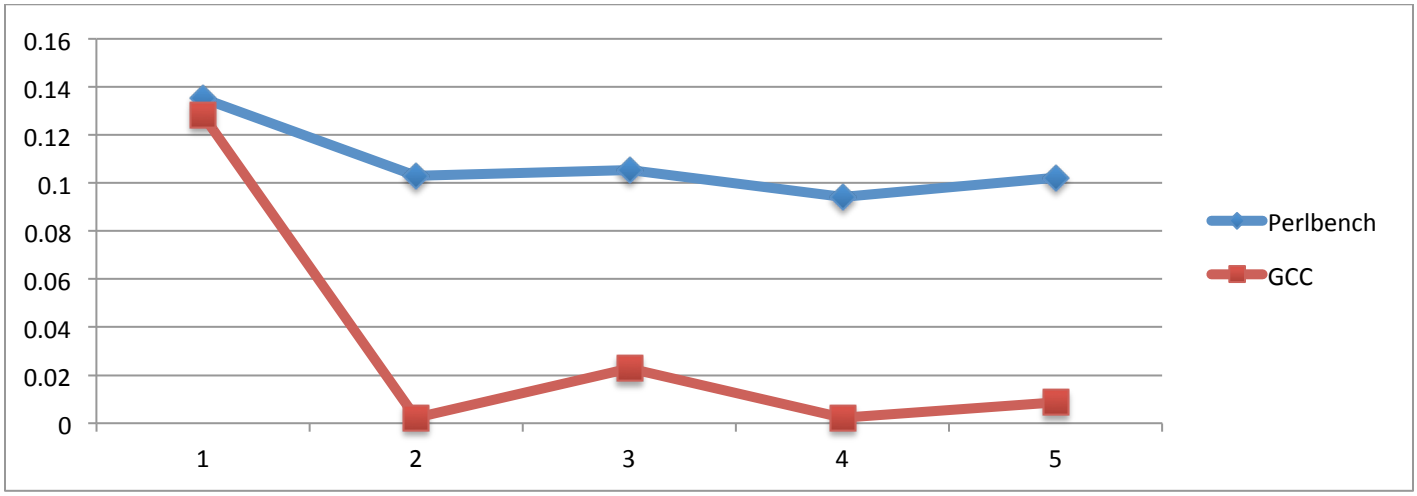
7. Changing number of entries affects predictors accuracy

Increasing the predictor accuracy doesn't have to improve prediction accuracy as the prediction performance depends on other parameters such as benchmarks, cache sizes etc.

8. Comparison of the performance of predictors with respective analysis and results.

	Predict Taken		**N-bit local (bimodal) branch predictor**						**Tournament predictor**	
	2048_2bits		**1024_1bit**		**1024_2bits**		**65536_2bits**			
Benchmark	PerlBench	GCC	PerlBench	GCC	PerlBench	GCC	PerlBench	GCC	PerlBench	GCC
IPC	0.483141	0.450264	0.544542	0.461711	0.539864	0.461697	0.564650	0.461713	0.558193	0.458170
Execution cycles	4357182	555230170	3865883	541464803	3901537	541480529	3730276	541462396	3773425	545648460
Branch rate	0.211846	0.227735	0.216964	0.075089	0.215175	0.075014	0.215390	0.075018	0.207691	0.077418
Branch misprediction rate	124864/ 923050 0.135273	16233573/ 12644509 0.1283843	86309 / 838248 0.10293	98769 / 40658266 0.00242	88576/ 839514 0.1055	92334/ 40618625 0.02273	75624/ 803464 0.094122	92479/ 40619505 0.00227	80135/ 783708 0.1022	367943/ 42242881 0.00871
% fetch squashed	488510/ 1333622 0.366303	78035732/ 18605696 0.4194185	371263 / 1142083 0.32507	775876 / 66254550 0.00117	376575 / 1145158 0.3288	613278/ 66084622 0.00928	332311/ 1085502 0.3061	615138 / 66089134 0.0093077	334649 / 1075491 0.3111	2214417/ 68829512 0.03217

Branch Misprediction:



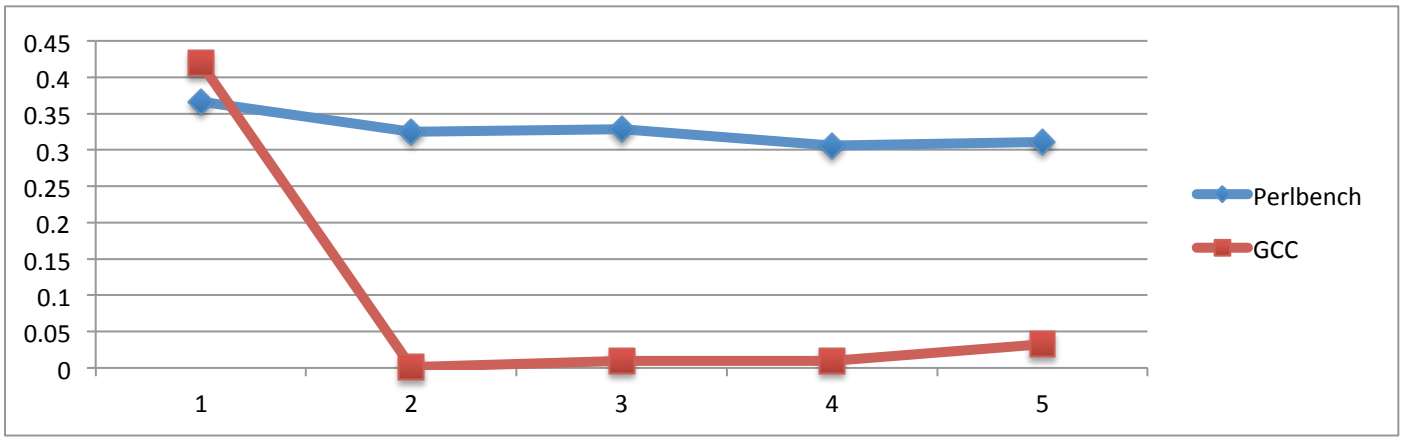
As the static predictors are less accurate than dynamic predictors, Predict taken has highest branch misprediction and percentage fetch squashed.

VIKRAM SRI NITESH TANTRAVAHI
ID: 13247361

In bimodal predictors, miss prediction rate increases with increased control bits and constant predictor size. With constant control bits, misprediction rate seems decreasing with the increase in predictor size. This means it has better prediction rate.

Tournament predictors have less misprediction rates as it uses both local and global predictors to predict and decide.

%Fetch squash cycle



Static predictors have highest percentage squash cycles, as they are less accurate than dynamic predictors.

In bimodal predictors, the case with higher counter bits has higher percentage squash rates thus miss rates are higher in this case. With varying predictor size and constant counter size, a smaller percentage squash rate is observed with lesser miss rate.

In tournament predictors, squash percentage seems very high. This is because they have multi-level branches increasing number of cycles and percentage fetch squash cycles.

Conclusion

Comparing the two workloads, GCC has lesser misprediction rate and fetch squash percentage than perlbench. Therefore it is optimum to choose GCC over Perlbench.