**UNIVERSITY OF TEXAS at DALLAS**

**DEPARTMENT OF ELECTRICAL ENGINEERING**

**CE 6304.0012 EEDG 6304**

**12/7/2016**

**Group Number 37**

**VENKATA KARTHEEK MADHAVARAPU**

**VIKRAM SHIVAPRAKASH**

**Contribution Percentage: Kartheek 50 % Vikram 50%**

**Project #2**

**“ANALYSIS OF DIFFERENT TYPES OF BRANCH PREDICTORS”**



**TABLE OF CONTENTS:**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Topic** | **Page Number** |
| **1** | Introduction | **3** |
| **2** | Part 1,Part 2 | **4** |
| **3** | Part 3 | **5** |
| **4** | Part 4 | **6** |
| **5** | Conclusion | **10** |

**OBJECTIVE:**

1. To discuss the effect of changing different branch predictors and what is the impact we observed by changing their parameters.
2. To discuss the challenges we faced when we used our environment and why we did not use the college server.
3. To show the result of our config.ini file which shows the TournamentBP.
4. To provide the changes we made to the source code in order to add the parameters required in Part 3.
5. To Generate the BTBMissPct and BranchMispredPercent for each of the Branch Predictors with the settings provided for each Benchmark.

**INTRODUCTION:**

In computer architecture, a branch predictor is a digital circuit that tries to speculate which way branch will go before this is known for sure (i.e , before its execution). The purpose of the branch predictor is to improve the flow in the instruction pipeline. They play a critical role in achieving high effective performance in many modern pipelined microprocessor architectures such as x86.

In this project, we analyze the behavior of different branch predictor configurations in five well-recognized benchmarks 401.bzip2, 429.mcf, 456.hmmer, 458.sjeng and 470.lbm

We have used three types of hardware based branch prediction strategies, they are:

1. 2bit\_local Predictor: A two level adaptive predictor with an n-bit history is that it can predict any repetitive sequence with any period if all n-bit sub-sequences are different.

2. Bi\_mode Predictor: It is a simple predictor, which uses 2-bit saturating counters to predict if a given branch is likely to be taken or not.

3. Tournament Predictor: A combined predictor implements more than one prediction mechanism. The final prediction is based either on a meta-predictor that remembers which of the predictors has made the best predictions in the past or a majority vote function based on an odd number of different predictors.

**PART 1:**

We are using our own environment. We are running it on virtual box and we have also dual booted our system into windows and Ubuntu.

Challenges faced: We did faced a few challenges during recompilation of Gem5, also we had to dual boot our system to Ubuntu which took a little time.

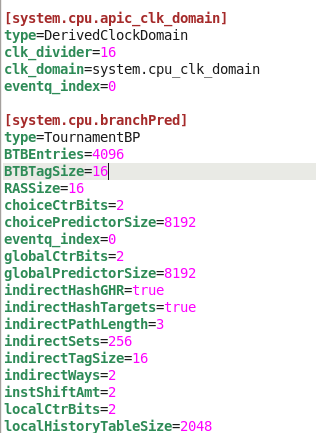
**PART 2:**

* By default the GEM5 TimingSImpleCPU does not have the BranchPredictor Support. We added the support of our Predictor.
* We followed the steps of adding the BranchPrediction support to Timing Simple CPU:

*$ce6304:> cd $gem5/src/cpu/simple*

* We edited the file named “BaseSimpleCPU.py” at the: branchPred = Param.BranchPredictor(NULL, "Branch Predictor")
* Changed the “NULL” to TournamentBP and then recompiled gem5.

**Below is the result of our config.ini file which shows the TournamentBP.**



**PART 3:**

**The following are the changes we made to the source file:**

1. *base.cc*

* t\_info.BranchMispredPercent

.name(thread\_str + ".BranchMispredPercent")

.desc("Number of branch mispredictions percent")

.prereq(t\_info.BranchMispredPercent);

* t\_info.BranchMispredPercent = (t\_info.numBranchMispred / t\_info.numBranches) \* 100;

1. *exce\_context.hh*

Stats::Formula BranchMispredPercent;

1. */pred/bpred\_unit.cc*

* BTBMissPct

.name(name() + ".BTBMissPct")

.desc("BTB Miss Percentage")

.precision(6);

* BTBMissPct = (1 - (BTBHits / BTBLookups) )\* 100;

1. */pred/bpred\_unit.hh*

Stats::Formula BTBMissPct;

1. *pred/Branchpredictor.py (Since we are using Gem5 and not the stable version)*

useIndirect = Param.Bool(False, "Use indirect branch predictor")

**PART 4:**

**Below is the tabular column where we have simulated 3 Branch Predictors with 5 benchmarks.**



In most of the benchmark simulations Tournament predictor appears to give better results. The tournament predictor inherently uses more than one predictor and a confidence/ meta predictor which decides the best result of the available results.

**PART 4(ii): We have chosen 2 bit Local BP to run on 401.bzip2 benchmark.**

**Below are the results of when the number of BTB entries is 512**

**Tabular column consisting of only BTBMissPct and BranchMispredPercent.**

|  |  |  |
| --- | --- | --- |
| **Size of Local Predictor** | **BTBMissPct** | **BranchMispredPercent** |
| 512 | 1.103787 | 6.896458 |
| 1024 | 1.082424 | 6.890964 |
| 2048 | 0.967376 | 6.889336 |
| 4096 | 0.900491 | 6.888324 |
| 8192 | 0.879941 | 6.888356 |

As we increase the size of the local predictor from 512 through 8192 keeping BTB entries at a constant 512 we notice that there is little improvement in the prediction percent in other words there is a less miss prediction percent for both the directions and the target addresses. However, the results prove that the improvement is very little on an average it is lesser than 0.1 percent. The other observation from the simulation results is that increasing the predictor size beyond 4096 is very less and can be expensive.

**Below are the results of when the number of BTB entries is 1024**

|  |  |  |
| --- | --- | --- |
| **Size of Local Predictor** | **BTBMissPct** | **BranchMispredPercent** |
| 2048 | 0.210449 | 6.599288 |

**Below are the results of when the number of BTB entries is 2048**

**Tabular column consisting of only BTBMissPct and BranchMispredPercent.**

|  |  |  |
| --- | --- | --- |
| Size of Local Predictor | BTBMissPct | BranchMispredPercent |
| 512 | 0.237376 | 6.603999 |
| 1024 | 0.216008 | 6.598527 |
| 2048 | 0.126096 | 6.594508 |
| 4096 | 0.125159 | 6.59062 |

As we increase the size of the local predictor from 512 through 4096 keeping BTB entries at a constant 2048 we notice that there is little improvement in the prediction percent in other words there is a less miss prediction percent for both the directions and the target addresses. However, the results prove that the improvement is very little on an average it is lesser than 0.05 percent.

**Below are the results of when the number of BTB entries is 4096**

**Tabular column consisting of only BTBMissPct and BranchMispredPercent .**

|  |  |  |
| --- | --- | --- |
| **Size of Local Predictor** | **BTBMissPct** | **BranchMispredPercent** |
| 512 | 0.203338 | 6.589619 |
| 1024 | 0.091989 | 6.580127 |
| 2048 | 0.091992 | 6.58013 |
| 4096 | 0.091054 | 6.576243 |
| 8192 | 0.07298 | 6.574446 |

As we increase the size of the local predictor from 512 through 8192 keeping BTB entries at a constant 4096 we notice a small improvement in the prediction percent of predictor size from 512 to 1024.

**Conclusion:**

**Tabular column of BTB 512,1024,2048,4096 for predictor size 2048.**

|  |  |  |
| --- | --- | --- |
| BTB | **BTBMissPct** | **BranchMispredPercent** |
| 512 | 0.967376 | 6.889336 |
| 1024 | 0.210449 | 6.599288 |
| 2048 | 0.126096 | 6.594508 |
| 4096 | 0.091992 | 6.58013 |
|  |  |  |

Below is the graph showing BTBMissPct and BranchMispredPercent of BTB 512,1024,2048,4096 for predictor size 2048.

From the graph we can infer that as we increase BTB size the branch direction prediction and target address prediction improving. However increasing the size of BTB is more expensive than increasing the size of the predictor so based on the improvement percentage and the size of BTB we have decided that BTB- 2048 and Local predictor size- 2048 would make the best combination.

*Thank You*