ANALYSIS OF BRNACH PREDICTORS

ABSTRACT

Goal of this paper is to analyze branch predictors and their importance in making a microprocessor. As the pipeline length increases, miss penalty increases accordingly. So, to minimize this miss penalty cycles we use branch predictors. More accurate the branch predictors more silicon area is required to implement it. Everything here in computer architecture is corelated but in this paper we will only see how branch predictors are helpful in making one.

BRANCH PREDICTIORS USED

TAKEN – This means that branch prediction is always taken and does not depend on any other things majorly.

NOTTAKEN – This means that branch predictions are always not taken, and it fetches next instructions immediately.

BIMOD - It implements a local branch predictor, which means that the effects of branches are localized. This means that branch predictions are handled individually, dependent on where the branch instruction is in the program. It uses a branch target buffer (BTB) and 2-bit saturating counters.

PERFECT – It predicts the more frequently executed outgoing edge of each branch in a program. If the perfect predictor has a low miss rate, then most branches follow one direction with high probability. If most branches take both directions with approximately equal probability, a perfect static predictor will do no better than a 50%miss rate.

2LEV - It implements a global branch predictor, which means that the effects of various branch instructions can influence the prediction. The branch behavior of the last 10 or so branches is kept in a global buffer. This value is then XORed with the lower bits of the address of the branch instruction to for an index. This index is then used to select an entry in the table.

COMB – It's a combination of bimod and 2lev predictors. Half of its entries are devoted towards one predictor and the rest toward the other. Selection of which predictor to use for a given branch is determined by the predictor depending on the complexity of predicting that branches instructions behavior.

REQUIREMENTS

SimpleScalar – to run simulation

BenchMark Programs – sample binaries to test

Shell Scripts – automate the task of testing multiple binaries.

ANALYSIS

I have considered Anagram, Compress and Go benchmarks for the analysis part of this paper and ran these binaries with many variations which you can see below.

One of my major observations in this process is branches executed is higher in taken prediction when compared others. For example, in any binary if the programmer only writes ‘if’ statement and does not include else for that then that branch is only counted as parsed only if the condition in that branch is true and this is always possible in ‘taken’ prediction. In case of ‘non taken prediction’ as the ‘if’ condition is not true, branch is not executed, and total count of branches executed in this type of prediction will be lesser than taken. This is one of the possibilities of why more branches are seen in results.

Another thing which we can say that if we provide more resources to CPU like register update unit and instruction fetch storage then CPI is lower in those cases which means programs run faster than before. These units help in updating register as well as fetch more instruction in a stretch.

So, predictions taken and not taken, are not the most sensible predictors which can be used because if we consider CPI values for all tested binaries. As the number of branches executed are high in both the cases, cycles needed to run this program will also be higher, so everything in this needed will be more like more power, time.

Considering bimod predictor, it will keep track of previously executed branch results and help it guess the next time it goes in that situation. This works well in any case and produces one of the top results in the tested predictors. Every conditional branch has a local buffer consisting of its entries and a history buffer with all available conditional jumps. So, CPI in this case is competitive when compared to others. It has BTB and predict whether it’s a conditional or unconditional branch.

In 2lev prediction, we have global buffer with conditional jumps up to certain limit and this is helpful in correlating branches across all conditional branches. Only disadvantage is that buffer gets diluted when branches are not related to each other and thus prediction does not make sense. Due to this it might not be good if the buffer is not huge.

Comb prediction is best prediction in the available prediction as it made of two or more predictors but in our case, it is a mixture of bimod and 2lev. As we have seen the disadvantage of 2lev previously now we can overcome and choose predictor based about the branch. Due to this added advantage, we get good results with this model.

These accuracies can be due to biased branches which always operate same way. And this can be easily handled by simple predictors.

Branch history length is one of the factors which are helpful in predictors, higher the lengths most accurate prediction takes place which is what I understood from the analyzing part and the research papers I have gone through.

Branch prediction is also related to the number of misses and hits There are values in the results of simple scalar where we can see how many addresses predicted hits for which shows out of available branches how many are successfully hit.

This value can also be used to see how our predictor is working with binaries. This ratio of hits to branches is very low for taken, not taken and moderate for other types. This means that others are performing well.

For comb predictor, we can see how many hits were made from considered types. For example, I have presented a sample of it.

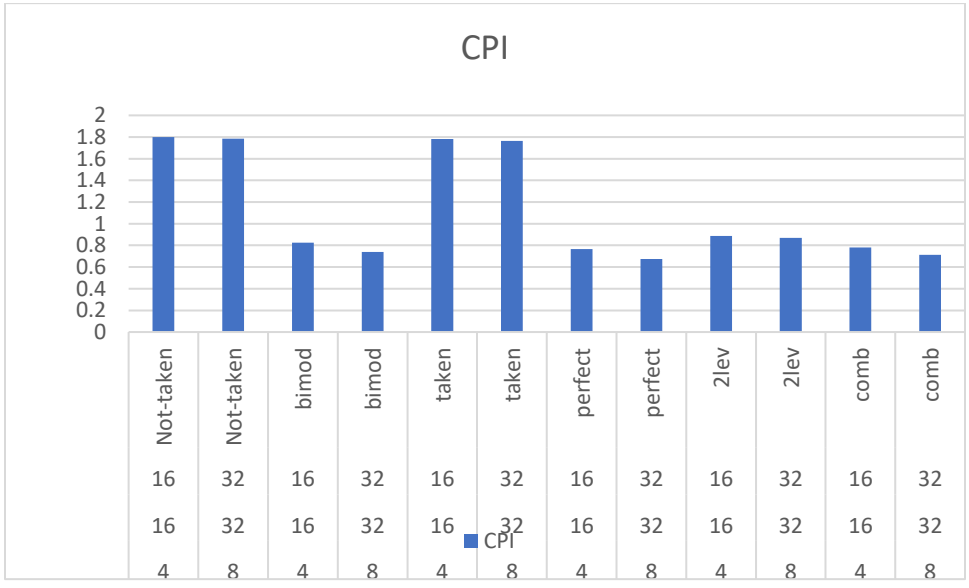
bpred_comb.used_bimod 6657600 # total number of bimodal predictions used

bpred_comb.used_2lev 2321686 # total number of 2-level predictions used

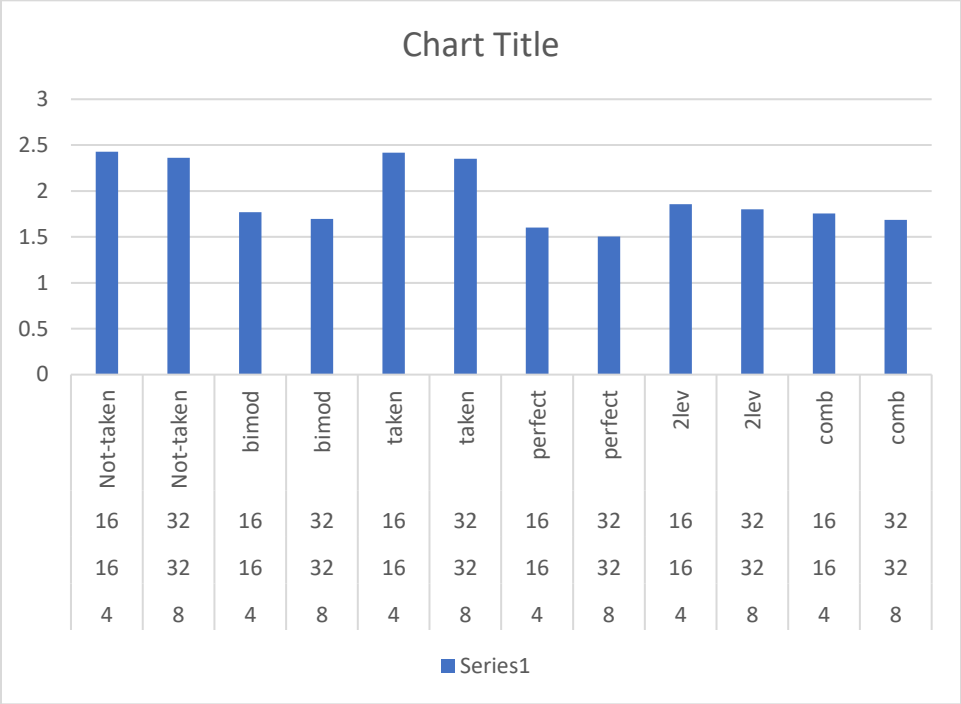
In all our observations, we can see that perfect was the top performer and there might be some reasons to it. For example if there are more for conditions in it, they always follow one direction and this makes prediction perfect and more accurate. If the directionality of branches is not uni direction the accuracy will degrade. So they might be many unconditional loops in every benchmark.

RESULTS

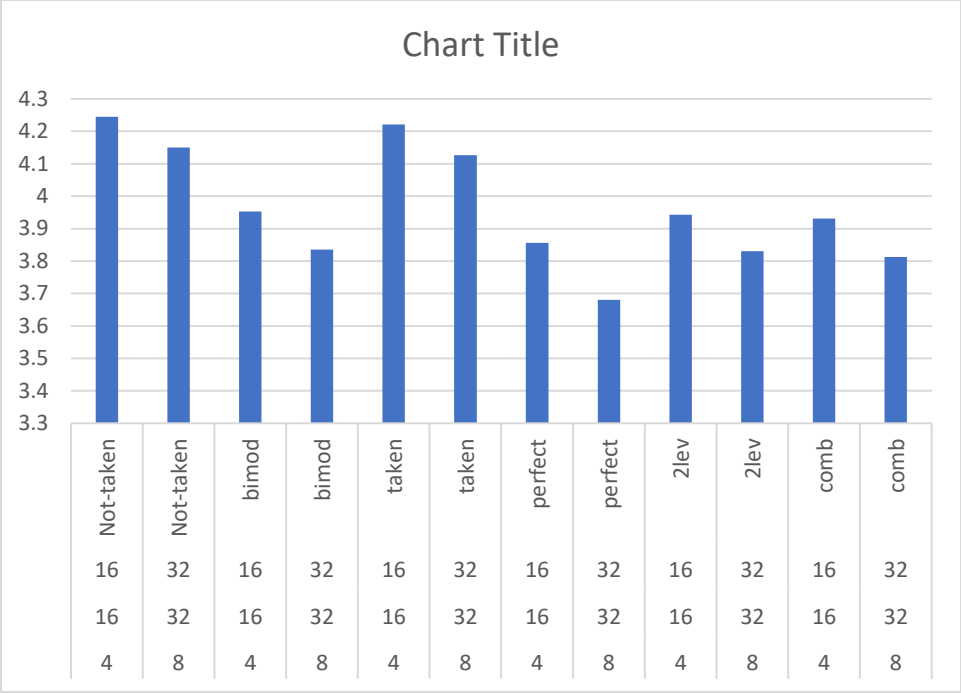
For Anagram:



For Compress:



For Go:



FUTURE WORK

we can try finding the number of conditional and unconditional branches using binary analysis. Also implement machine learning algorithms for more efficient correlating of history which takes 2lev and bimod to another stage of prediction.

CONCLUSION

For the considered benchmark, due to mentioned reasons above perfect and comb predictors worked very well.

REFERENCES

final_paper_new (emulationzone.org)

18-741 Advanced Computer Architecture Lecture 1: Intro and Basics (cmu.edu)

(PDF) Branch Classification: a New Mechanism for Improving Branch Predictor Performance (researchgate.net)