CS/ ECE-570 HPCA: Final Project Report

Eric Prather
Department of EECS
Oregon State University
Corvallis, OR
Email: prathere@oregonstate.edu

Si Thu Lin
Department of EECS
Oregon State University
Corvallis, OR
Email: linsi@oregonstate.edu

Dhruv Jawalkar
Department of EECS
Oregon State University
Corvallis, OR
Email: jawalkad@oregonstate.edu

***Abstract*—The purpose of this project is to gain understanding of branch prediction and explore its design space. Computer architects in industry use software-based microarchitectural simulation tools to analyze bottlenecks, rapid prototyping of new ideas, and to compare the relative merits of different ideas.**

***Index terms*—** Keywords: SimpleScalar, l1 size, l2 size, GO, GCC, ANAGRAM

I. Introduction

In this project we will use an existing out-of-order superscalar processor simulator called SimpleScalar as the simulation tool. This simulator, like many other processor simulators, reads a machine configuration file and generates a processor model that matches the requirements specified in the configuration file. The configuration file specifies parameters such as the size and associativity of caches, the size of RoB, the number of functional units etc. This project is meant to become familiar with branch Prediction schemes and their design spaces. The task would be to use "sim-outorder" to run different benchmarks for different branch prediction schemes and note their results to compare dif- ferent parameters. Different configurations are tested on GCC, GO and ANAGRAM benchmarks.

The remainder of the report is arranged as follows. Section-2 compares the Address-Prediction Rate obtained for different branch prediction schemes on the aforementioned benchmarks. Section-3 will compare their Instructions Per Cycle (IPC). Section-4 will compare the Address-Prediction Rate for different table entry numbers for bimodal branch predictor.

II. Address-Prediction Rate

A. Execute benchmarks for various Branch Predictors

We ran "sim-outorder" executable for five different branch prediction schemes on all the benchmarks. Table-1 contains the values of Address-Prediction Rate for all those schemes.

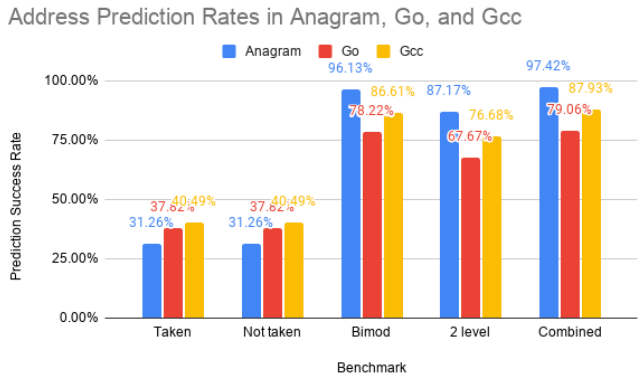
Figure-1(a) shows the histogram of Address-Prediction Rates for all the three benchmarks. ANAGRAM being a simpler benchmark than GCC and GO provides very high accuracy for all the branch predictor schemes. GO being the most complicated benchmark among the three, has the worst prediction accuracy.

TABLE I
ADDRESS-PREDICTION RATES

Benchmark	Branch Prediction Schemes				
	Taken	Not taken	Bimod	2 level	Combined
Anagram	0.3126	0.3126	0.9613	0.8717	0.9742
Go	0.3782	0.3782	0.7822	0.6766	0.7906
GCC	0.4049	0.4049	0.8661	0.7667	0.8793

The patterns as seen in the histogram show that the "combined" branch prediction scheme provides the most accurate ad- dress predictions, whereas "Taken" and "Not taken" being static predictors, provide very poor accuracy. Also, the Address prediction rate is same for both "taken" and "not taken" schemes, this can be because the benchmarks must be having equal number of branch instructions in both the directions. Since "Bimod" branch prediction scheme is the 2-bit predictor, as observed it has a very good accuracy on almost all benchmarks, second to only the "Combined" prediction scheme. Almost all the observations agree with the intuition regarding the expected results. As expected, the accuracy of both the static approaches, "taken" and "not-taken" is the worst of all. This is because, for every branch instruction, the decision for both the algorithms is same irrespective of the history of the branch or the correlation of the branch with the previous branch instructions. For "Combined" prediction scheme, the best results were expected. This is be- cause combined strategy is the combination of "Bimod" branch predictor and the "2 level" branch prediction scheme. Thus, it was expected to perform better for situations where either one of its parents' mis-predicts. One thing that did not follow our expectations was the accuracy difference between the 2-level and the bimod predictions. Bimodal predictor, is a simple branch prediction method which uses the address of the branch instruction to index the global table of prediction bits. Two-level branch prediction scheme is an adaptive approach. It also keeps track of the correlation between the current branch and the previous branches. Thus, this scheme also consists of making use of a history table. Therefore, we expected "two-level" to per- form better than "bimod" branch predictor, this is not the case for all the benchmarks. This aligns with the findings of Soundararajan et al., who indicated that the state of the art of dynamic two-level branch predictors are not yet optimized to outperform

their counterparts.[1] We believe that the benchmarks might be having some "hard to predict" branch that might have messed up the history table for two-level predictor, leading to more mispredictions. The above-mentioned logic also agrees with the observations.



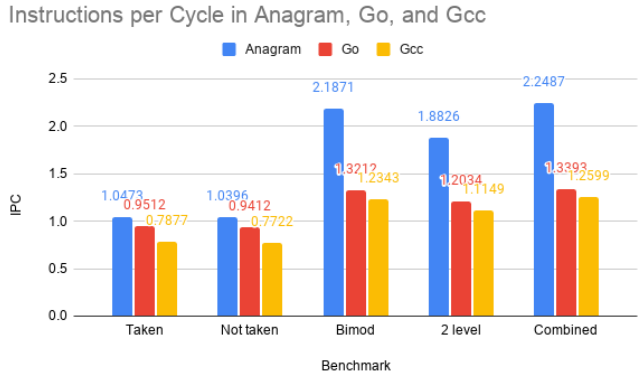


Fig. 1. (a) Address-Prediction rate vs Branch Prediction Type (b) Instructions Per Cycle vs Branch Prediction Type

III. Instructions per cycle (IPC)

A. Execute benchmarks for various Branch Predictors

TABLE II
INSTRUCTIONS PER CYCLE

Benchmark	Branch Prediction Schemes				
	Taken	Not taken	Bimod	2 level	Combined
Anagram	1.0473	1.0396	2.1871	1.8826	2.2487
Go	0.9512	0.9412	1.3212	1.2035	1.3393
GCC	0.7878	0.7722	1.2342	1.1147	1.2598

[1]Niranjan Soundararajan et al. "Towards the adoption of Local Branch Predictors in Modern Out-of-Order Superscalar Processors". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 519–530. ISBN: 978-1-4503-6938-1. DOI: 10.1145/3352460.3358315. URL: https://doi.org/10.1145/3352460.3358315 (visited on 11/14/2020), p. 1.

TABLE III
IPC AVERAGES

Test	Taken	Not taken	Bi-mod	2 lvl	Comb	Best
Arith. Mean	0.929	0.918	1.581	1.400	1.616	Comb
Geo. Mean	0.922	0.911	1.528	1.362	1.560	Comb
Harm. Mean	0.916	0.904	1.482	1.328	1.511	Comb

We ran "sim-outorder" executable for five different branch prediction schemes on all the benchmarks. Table-2 contains the values of IPC for all those schemes.

Figure-1(b) shows the histogram of Instructions Per Cycle (IPCs) for varying predictor types for all three benchmarks. ANAGRAM being a simpler benchmark than GCC and GO provides better IPC for all the branch predictor schemes. GO being the most complicated benchmark among the three, has the lowest IPC.

As expected, the worst performing branch predictors are the "Taken" and "Not Taken" predictors. This is because they follow the simplest branch prediction technique and does not rely on information about the dynamic history of code execution. The "two-level" and "Bimodal" branch predictors are able to achieve significantly better performance in comparison to the static branch predictors. Also, the Bimodal predictor performs slightly better than the 2-level predictor. This difference in performance is due to the dynamic nature of the predictors. The Combined branch predictor, which is the combination of the above branch predictors, has the best performance in all three benchmarks.

IV. Address-Prediction rate for different Bimodal Predictor Table Entry Numbers

A. Execute benchmarks for various table entry numbers for Bimodal Branch Predictor

TABLE IV
ADDRESS PREDICTION RATES FOR BIMOD

Benchmark	Table Entry Numbers				
	256	512	1024	2048	4096
Anagram	0.9606	0.9696	0.9612	0.9613	0.9613
Go	0.743	0.761	0.7731	0.7822	0.7885
GCC	0.8158	0.8371	0.8554	0.8661	0.8726

We ran "sim-outorder" executable for five different number of table entries for "Bimod" Branch Prediction scheme on all the benchmarks. Table-3 contains the values of Address-Prediction Rate for all those sizes.

Figure-2 shows the histogram of Address-Prediction Rates for all the three benchmarks. The Address-Prediction rate is

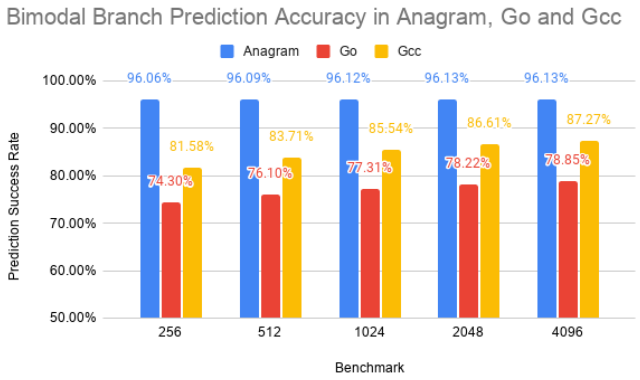


Fig. 2. Bimodal Branch Prediction Accuracy

directly proportional to the number predictor table entries, i.e., as the size of the branch target buffer (BTB) increases, there is an increase in the address prediction rate.

ANAGRAM being a simpler benchmark than GCC and GO has almost similar prediction rate for all the sizes. GO being the most complicated benchmark among the three, show increase in prediction rates as the size of table entry increases. Although, there is a very slight increment.

V. WORKING OF COMBINED BRANCH PREDICTOR

TABLE V
SELECTION COUNTER

Predictor-1	Predictor-2	Update to counter
0	0	No change
0	1	decrement
1	0	increment
1	1	No change

There are different branch predictors with varying advantages. One effective way is to combine any two branch prediction schemes to form a new branch prediction with better prediction accuracy. The two branch predictors are basically, a local branch predictor which can predict branches like inner loops and a global branch predictor which can predict branches like correlated branches. These two branch predictors work independently from each other and one of the best way is to combine both these predictors in one and a selector is used to choose between the two predictors. The selector, Table-4, is basically a two bit up/down saturating counter which keeps track of which predictor is more accurate for each branch. The selection counter is updated only if the prediction given by the two predictors is different. Each branch is mapped into its counter via its address. The counter is incremented based on the table shown below. The most significant bit of the counter determines which one of the two predictors to use.

These branch predictors have better accuracy compared to other predictors and reduced warmup time. Of the branch predictors given the combination of bimodal and 2-level gshare gives better performance than other combinations because of the warmup time.

As we the see from the above tabulation, the combined branch predictor has a better predictor rate in terms of address prediction rate and instruction per cycle. In this project we have considered the combination of bimod and 2-level adaptive branch predictor (GAp). If only the bimodal predictor gives a correct prediction, the counter is decremented. The counter is incremented if only the 2-level structure gives a correct prediction. If both predictors either give a correct or an incorrect prediction, the selection counter is not updated. Consequently, state 0 and 1 of the state machine entails a selection of the bimodal predictor, while state 2 and 3 result in the use of the 2-level.

VI. 3 BIT BRANCH PREDICTOR

We implemented a 3-bit branch predictor. The results produced by this predictor are in table 5. The implementation is available on GitHub.

TABLE VI
3-BIT BRANCH PREDICTOR PREDICTION ACCURACIES IN VARIOUS BENCHMARKS

Benchmark	Table Entry Numbers				
	256	512	1024	2048	4096
Anagram	0.9610	0.9612	0.9617	0.9617	0.9617
Go	0.7507	0.7680	0.7799	0.7899	0.7969
GCC	0.8191	0.8385	0.8555	0.8657	0.8728

We compare the 3-bit branch predictor to the two-bit branch predictor in terms of their performances in Table 6 and Figure 3.

TABLE VII
2-BIT VS. 3-BIT PERFORMANCES

Size	Table Entry Numbers				
	256	512	1024	2048	4096
Anagram (2-bit)	96.06	96.09	96.12	96.13	96.13
Anagram (3-bit)	96.10	96.12	96.17	96.17	96.17
Go (2-bit)	74.30	76.10	77.31	78.22	78.85
Go (3-bit)	75.07	76.80	77.99	78.99	79.69
GCC (2-bit)	81.58	83.71	85.54	86.61	87.27
GCC (3-bit)	81.91	83.85	85.55	86.57	87.28

Explanation for the performance increase Any Bimodal Branch Predictor cannot outperform 2-bit Branch Predictor in terms of making predictions for the loop branch regardless of how many bits are used for the counters. However, using many bits for the counters can increase the confidence of the predictor when it makes predictions for the if-then-else branch. Small table sizes give less performance. When the table size is small, more than one instruction can share the same

entry of the counter table,which can decrease the accuracy of predictions. Therefore, the larger the table size is, the better the performance is.

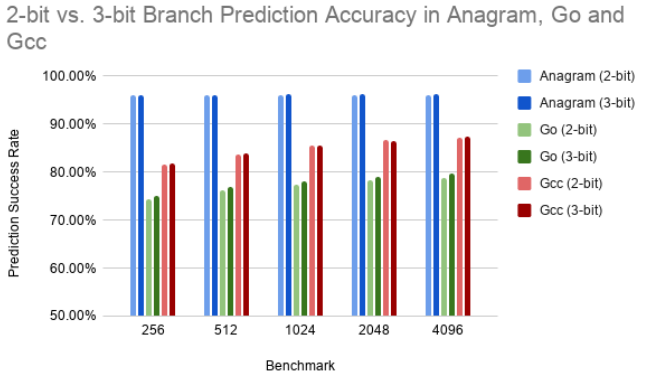


Fig. 3. Side-by side comparison of 2-bit and 3-bit prediction

We determined, contrary to our expectations, that the 3-bit predictor did not always perform substantially better than its 2-bit counterpart. This is evidenced in the 256-sized run of the anagram benchmark, where the 3-bit predictor slightly outperformed the 2-bit one by a slim margin of just 0.03. Because this simulation is deterministic, we cannot attribute this to random error, although we may explain the fact that the magnitude of the improvement is inversely proportional to the base prediction accuracy, as Anagram was already well over 90 accurate when bimodal prediction was applied to it. Alternatively, we could conclude that in the anagram benchmark, there are many cases where two mispredictions signify a change in the pattern of the behavior, which is ignored by the 3-bit predictor but acknowledged by the 2-bit predictor.

Aside from this case, however, the 3-bit predictor universally outperforms the 2-bit predictor. This is particularly notable in the Go benchmark, although the effect can be observed in all of them.

REFERENCES

[1] Web.engr.oregonstate.edu. (2019). Dynamic Branch Prediction. [online] Available at: http://web.engr.oregonstate.edu/~benl/Projects/branch_pred/

[2] D. Burger and T. M. Austin, “The SimpleScalar tool set, version 2.0,” ACM SIGARCH Computer Architecture News, vol. 25, no. 3, pp. 13–25, 1997. http://www.simplescalar.com/docs/users_guide_v2.pdf

[3] D. A. Jiménez, “Code placement for improving dynamic branch prediction accuracy,” ACM SIGPLAN Notices, vol. 40, no. 6, p. 107, 2005

[4] M. Dubois, M. Annavaram, and P. Stenstrom, “Parallel Computer Organization and Design,” 2009