ECE570 Final Project Report

Faaiq Waqar
School of EECS
Oregon State University
Corvallis, OR
waqarf@oregonstate.edu

Braam Beresford
School of EECS
Oregon State University
Corvallis, OR
beresfob@oregonstate.edu

Stephen More
School of EECS
Oregon State University
Corvallis, OR
mores@oregonstate.edu

I. INTRODUCTION

This document outlines procedure and findings of the high performance computer architecture project. The goal of this project is to gain an understanding of branch prediction designs in a simulated environment, with use of the SimpleScalar [1] CPU simulator program. All benchmarks highlighted in this report are done through an out of order processor simulator, and have been bench marked on the Ubuntu 12.04 operating system.

II. CROSS-METHOD BENCHMARKS

The following section goes through initial benchmarks on the vanilla system. The set of branch prediction models used are taken, not taken, bimodal, 2 level and combined predictors. These are tested on the anagram, go and gcc compilers, across a number of different table sizes.

A. Address-Prediction Rate

TABLE I
CROSS-PREDICTOR ADDRESS-PREDICTION RATE

Benchmark	***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	0.6767	0.7906
gcc	0.4049	0.4049	0.8661	0.7668	0.8793

Our intuition was that the combined, 2 level and bimod branch predictor (comb) would have the highest hit rate across benchmarks. The thought process behind this is that by leveraging advantages in each branch prediction methodology, across a range of benchmarks, the combination should have the highest hit rate. This intuition was validated by our data at Figure 1, which shows that the comb branch prediction has a higher hit rate than any other.

B. Instructions per Cycle

The combined predictor also has the greatest number of instructions per cycle for all benchmarks. This aligns with the previous table as when there is the highest rate of branch prediction hit rate, there is less processor time being wasted by correcting mistakes in branch prediction. When less time is wasted more instructions are able to be executed per cycle. The resulting histogram for IPC is shown at Figure 2.

Identify applicable funding agency here. If none, delete this.

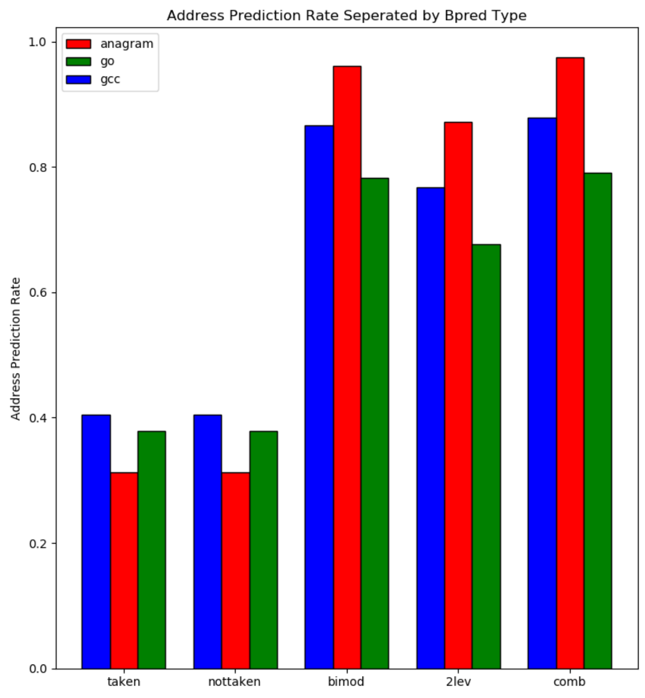

Fig. 1. Address prediction rates across benchmarks and branch prediction types

TABLE II
CROSS-PREDICTOR INSTRUCTIONS PER CYCLE

Benchmark	***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	1.2034	1.3393
gcc	0.7877	0.7722	1.2343	1.1149	1.2599

III. BIMODAL PREDICTOR BENCHMARKS

As the number of branch prediction buffer (BPB) entries increase (shown in Figure 3), there are more FSMs allocated to branch target address space. The FSMs are mapped to memory with direct mapping. This means that as we increase the number of entries, performance loss related to replacement misses are reduced. However, the relationship is not linear, similar to increasing cache size in direct mapped cache, there is a ”sweet spot” in which we get the greatest performance increase with least amount of entries.

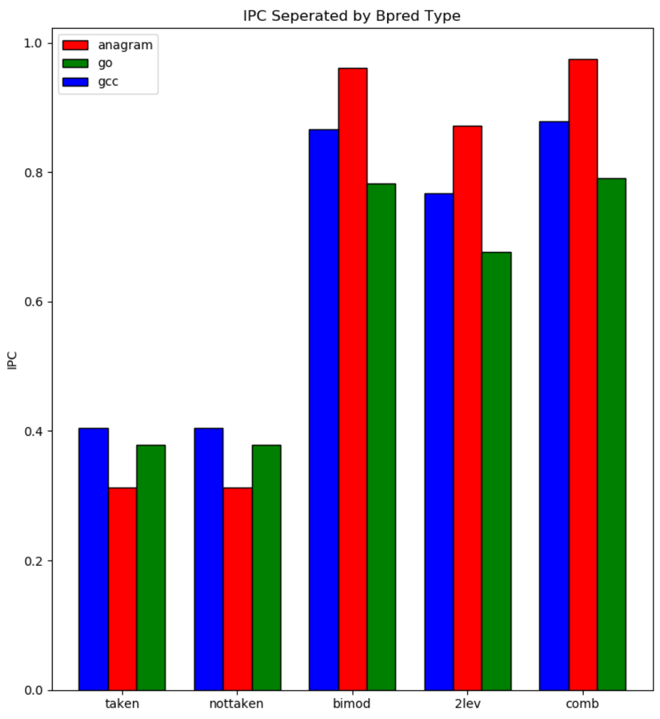

Fig. 2. Instructions per cycle across benchmarks and branch prediction types

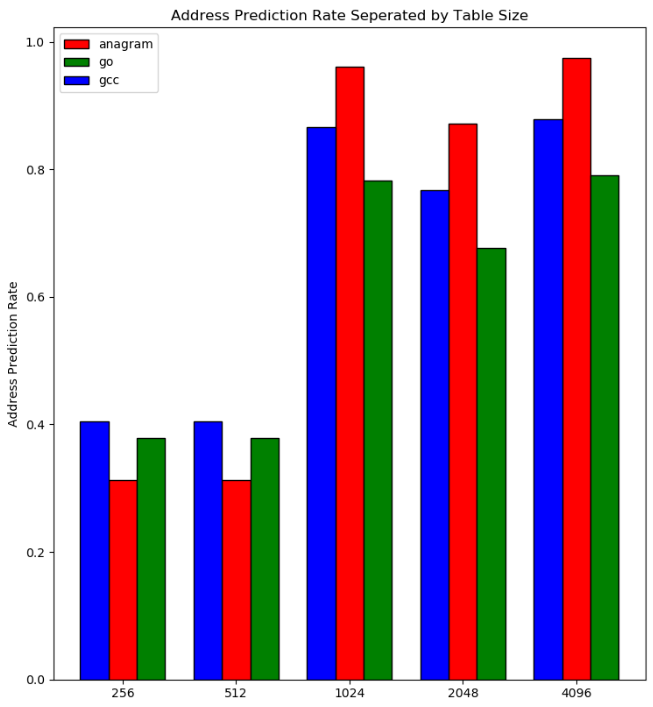

Fig. 3. Address prediction rates of bimod predictor across benchmarks and bimod table sizes

TABLE III
BIMODAL ADDRESS-PREDICTION RATE BENCHMARKS

Benchmark	***256***	***512***	***1024***	***2048***	***4096***
anagram	0.9606	0.9609	0.9612	0.9613	0.9613
go	0.7430	0.7610	0.7731	0.7822	0.7885
gcc	0.8158	0.8371	0.8554	0.8661	0.8727

IV. COMBINED BRANCH PREDICTOR ARCHITECTURE

In pursuit of increasing instruction level parallelism (ILP), the overhead created by conditional branching direction and target address calculation remained a major challenge through development early computer systems. Literature contains many proposals for the use of branch prediction designs that seek to maximize prediction accuracy. Some methods are static, and rely on information on instruction type (by opcode) and profiling statistics to make predictions [2], while others run on algorithms rely run-time statistics to reach a branching decision. These algorithms, called static and dynamic branch predictors respectively, give us the basis for the combined branch predictor. [4]

With the success of global and local branch predictor schemes, Scott McFarling of the Western Research Laboratory proposed a method of combining the advantages of two arbitrary branch prediction schemes that becomes greater than the sum of its parts. [4] The following section discusses the foundations for understanding the design, implementation and use of the McFarling's combined branch predictor conceptually and in SimpleScalar. [1]

A. Combining Branch Predictors

The combined branch predictor is less about joining two very specific branch predictors. Rather, McFarling's predictor provides intuition as two how to reap the benefits of the most viable attributes of any two branch predictors. In SimpleScalar, the implementation works off the back of the bimodal branch predictor and the two-level branch predictor.

The design of combined branch predictor is composed of three elements. The first are the two branch predictors that are being combined. These, for simplicity's sake, are referred to as *P1* and *P2*. *P1* and *P2* are indexed by program counter (*PC*). The second necessary element is a combined predictor table, which is also indexed by *PC* bits. The final element is a 2-bit saturating counter used to serve the selection of the the most accurate direction of the two branch predictors. This counter, designed similarly to the finite state machine (FSM) used in a bimodal predictor, exists as an instance in each table element, tying it to a subset of branches. [2] A visual representation of the FSM used by the combined predictor can be seen in Figure 4, where the transition function is determined by *P1c*-*P2c*. The notation *P1c* and *P2c* denote the correctness of *P1* and *P2* respectively. [4]. The MSB is used to determine the predictor used, with an MSB of 0 indicating a bias towards P2, and an MSB of 1 indicating a bias towards P1.

Table IV shows the computation process of the transition function used in the combined branch predictor's saturation counter. The counter saturates at 3, and never drops below 0. Correctness and incorrectness in both predictors simultaneously result in no change in the saturating counter.

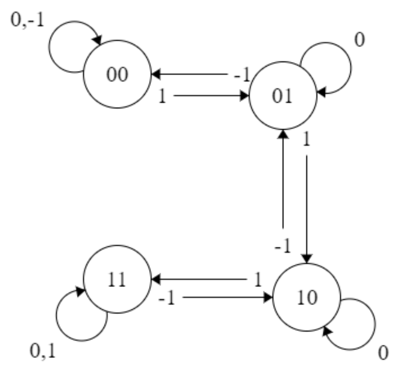

Fig. 4. Finite State Machine used for Combined Branch Predictor.

TABLE IV
COMBINED PREDICTOR COUNTER LOGIC

P1c	*P2c*	*P1c-P2c*	*Change Made*
0	0	0	No Change
0	1	-1	Decrement Counter
1	0	1	Increment Counter
1	1	0	No Change

The design of this combined branch predictor in its entirety is shown in figure 5. While more computationally expensive, the use of combining branch predictors can result in more accurate prediction of branch direction.

B. Use in SimpleScalar

In order to make use of the combined branch predictor in SimpleScalar, one must use the following flag to set the combined branch predictor.

-bpred comb

One must also set the table size for the combined branch predictor using the following flag. Size must be a multiple of 2. By default, the combined branch predictor table size is set to 1024. [1]

-bpred:comb [size]

C. Design in Simplescalar

The SimpleScalar implementation of the combined branch predictor defaults to the combination of the bimodal branch predictor proposed by Lee and Smith [3] and the two-level adaptive branch predictor proposed by Yeh and Pat. [5]. These papers provide a stronger insight into the design and methodology of each predictor. It is important to note that both designs focus on dynamic branch prediction, each having its own strengths and weaknesses. Using the combined predictor reaps the benefits of both predictors.

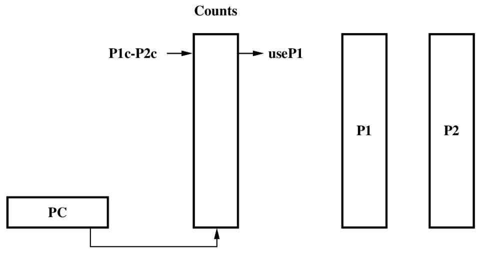

Fig. 5. Design of Combined Branch Predictor [4]

V. 3-BIT BRANCH PREDICTION

TABLE V
3-BIT ADDRESS-PREDICTION RATE BENCHMARKS

Benchmark	***256***	***512***	***1024***	***2048***	***4096***
anagram	0.9610	0.9612	0.9615	0.9616	0.9616
go	0.7507	0.7680	0.7799	0.7897	0.7966
gcc	0.8191	0.8385	0.8555	0.8657	0.8728

The 3-Bit implementation of a saturating branch predictor saw improvements across the board compared to the 2-Bit implementation. Despite this it was interesting this the design saw some similar limitations when it came to scaling of performance with the increase in table entries. Similar to 2-Bit, the three bit saw a plateau of improvements at 2048 instructions for the anagram program likely due to it being the shortest of the programs and therefore not getting any more replacement misses after 2048.

The go and gcc tests had continuous increases in performance with table size scaling although diminishing returns with table size. Comparatively go had the lowest prediction rate followed by gcc and then anagram; this is a result of program complexity. The go program was by the far the longest executing which leads us to believe that the programs complexity combined with the 3-Bit predictors limited ability to predict complex taken/not-taken sequences results in the lowest performance.

Overall the 3-Bit predictor is an improvement over 2-Bit as it is more robust at keeping with a long term trend of predictions over periodic deviations from that trend. We predict that scaling the number of bits up will have diminishing returns to the point where it will begin to reduce performance as the predictor will continuously lag behind changes in the execution pattern.

REFERENCES

[1] D.Burger, T. Austin. ”The SimpleScalar tool set, version 2.0 ” in SIGARCH Comput. Archit. News 25, 3 (June 1997), 13–25. doi: 10.1145/268806.268810

[2] N. Bellas, I. Hajj and C. Polychronopoulos, ”Using dynamic cache management techniques to reduce energy in a high-performance processor,” Proceedings. 1999 International Symposium on Low Power Electronics and Design (Cat. No.99TH8477), San Diego, CA, USA, 1999, pp. 64-69, doi: 10.1145/313817.313856.

[3] J. Lee, A. Smith, ”Branch Prediction Strategies and Branch Target Buffer Design,” in Computer, vol. 17, no. 1, pp. 6-22, Jan. 1984, doi: 10.1109/MC.1984.1658927.

[4] S. McFarling, “Combining Branch Predictors,” TR, Digital Western Research Laboratory, JUN 1997.

[5] T.Yeh, Y. Patt, ”Two-level adaptive training branch prediction” in Proceedings of the 24th annual international symposium on Microarchitecture (MICRO 24). pp. 51–61. 1991, doi: 10.1145/123465.123475