CS570: Final Project – Branch Prediction

Chendi Zhang
zhangch3@oregonstate.edu

Zhuoling Chen
chenz6@oregonstate.edu

Xintong Wang
wangxint@oregonsate.edu

Introduction

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

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.

Tasks and Requirements

This project will let you become familiar with branch prediction schemes and their design spaces. You will use sim-outorder to run different benchmarks. The guidance of the SimpleScalar can be found at link http://www.simplescalar.com/docs/users_guide_v2.pdf.

It will resolve the most of problems during testing. ./sim-outorder -h can also give you some helpful information.

I. Basic Benchmark Prediction

Benchmarks:

- The following tasks all need to run 3 benchmarks included in the downloaded benchmark folder: GCC, ANAGRAM (input data: anagram.in), GO (input config: 50 9 2stone9.in)
- The instructions of running benchmarks are included in the benchmark folder
- Use Alpha Benchmark

A. *address-prediction rate for different branch pridictors*

Execute Benchmarks and fill Table 1 with address-prediction rate and plot a histogram for all the three benchmarks(Fig. 1).

Conclusion:

1) The taken and no-taken predictions rate are the same.
2) Dynamic predictors performance are better than the static predictors.
3) The performance of Bimod predictor is better than 2 level predictor.
4) Combined predictions performance are better than the Bimod and 2 level separately.

Does your conclusion agree with your intuition? Actually the third conclusion is different with what we thought, we thought 2 level predictor should be better because the structure is actually different.

TABLE I
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	0.6768	0.7906
gcc	0.4049	0.4049	0.8661	0.7668	0.8793

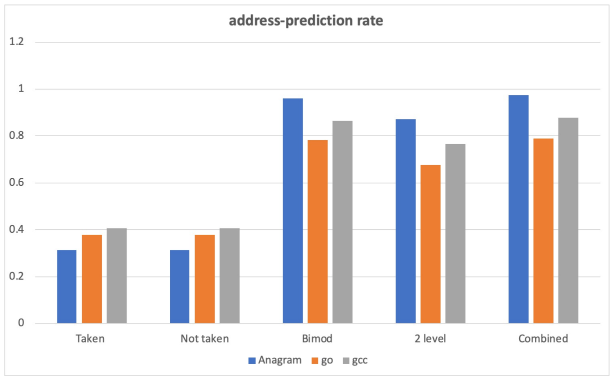


Fig. 1. address-prediction rate

B. *IPC for different branch predictors*

Execute Benchmarks and fill Table 2 with IPC and then plot a histogram for all the three benchmarks(Fig. 2).

TABLE II
IPC

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	1.2035	1.3393
gcc	0.7877	0.7722	1.2343	1.1148	1.2598

C. *address prediction rate for different cache size*

Use bimodal branch predictor, change the bimodal predictor table entry numbers, execute Benchmarks and fill Table 3 with

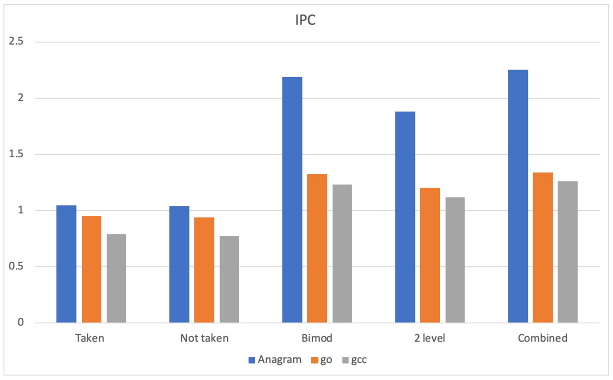


Fig. 2. IPC

address-prediction rate and then plot a histogram for all the three benchmarks(Fig. 3).

TABLE III
ADDRESS PREDICTION RATE

Benchmark	256 b	512 b	1024 b	2048 b	4096 b
Anagram	0.9606	0.9606	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	0.8726

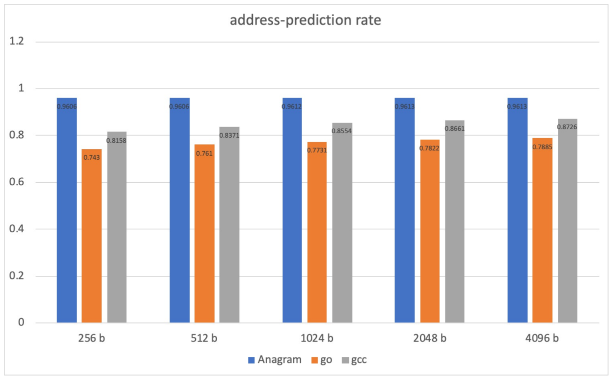


Fig. 3. address-prediction rate

II. Describe how combined branch predictor works

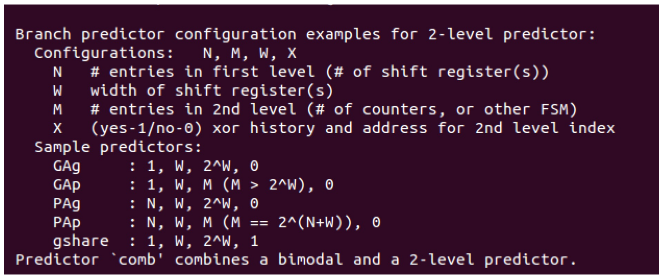
The combined branch predictor we use is to have 3 different branch predictors and merge their results by a majority vote. Predictors use multiple table entries to track the behavior of any particular branch. This multiplication of entries makes it much more likely that two branches will map to the same table entry (a situation called aliasing), which in turn makes it much more likely that prediction accuracy will suffer for those branches.

Once you have multiple predictors, it is beneficial to arrange that each predictor will have different aliasing patterns, so that it is more likely that at least one predictor will have no aliasing.

The basic method of combine branch predictors are:

- GAg : Branch history is global with Hist Size W Pattern history is addressed with W bits
- GAp: Branch history is global with Hist Size W Pattern history is addressed with those W bits+extra PC LSB bits. Adds semblance of Locality
- PAg: Separate Branch History for N addresses. log2N bits of PC taken. Branch history of each address, W bits. Only uses Branch history to address Pattern History
- PAp: Separate Branch History for N addresses. log2N bits of PC taken.Branch history of each address, W bits. Combines LSBs of PC and Branch History to address Pattern History
- Gshare: Global History Register, Global History Table with Simple attempt at anti-aliasing

The Fig. 4 is showing the relations between parameters in different branch predictor.



```
Branch predictor configuration examples for 2-level predictor:
  Configurations:   N, M, W, X
    N   # entries in first level (# of shift register(s))
    W   width of shift register(s)
    M   # entries in 2nd level (# of counters, or other FSM)
    X   (yes-1/no-0) xor history and address for 2nd level index
  Sample predictors:
    GAg     : 1, W, 2^W, 0
    GAp     : 1, W, M (M > 2^W), 0
    PAg     : N, W, 2^W, 0
    PAp     : N, W, M (M == 2^(N+W)), 0
    gshare  : 1, W, 2^W, 1
Predictor `comb' combines a bimodal and a 2-level predictor.
```

Fig. 4. different combine branch predictors

III. Find the best two-level branch predictor

A. *description*

In SimpleScalar, it can simulate five 2-level branch predictor configurations: GAg, GAp, PAg, PAp, gshare. Your task is to find the best 2-level branch predictor in terms of performance by changing parameters without exceeding storage constraints. You can change only four parameters of each 2-level branch predictor: l1_size, l2_size, hist_size, and XOR. The performance is defined as follow:

Performance evaluation metric: Average IPC = (IPCGCC + IPCANAGRAM + IPCGO) / 3

Please make sure the storage consumption of the predictor (storage consumption =l1_size * hist_szie + l2_size * 2, unit: bits) does not exceed 2 KB. More details can be found in the user's guide. Designs that violate this constraint will receive 0 points on this part.

B. *research*

We studied for those 5 methods and ran several different combination of the L1, L2,Hist and XOR parameters.

We run several test cases of each method, then we found out that the performance of GAg and GAp are clearly worse than other branch predictors.

We also run some test cases for PAg, PAp and gshare, found out that the larger of L2 size is the better performance the predictor have. Therefore, we selected 3 particular cases and shows the results in the tables below. We also choose one group of parameters with 64 L2 because we found the performance is not bad.

The Table IV shows the parameters and the result(rate) of these selected groups.

TABLE IV
RESEARCH PREDICTION RESULT

L1	L2	hit	XOR	Anagram	go	gcc	AVG
64	512	9	0	2.1972	1.1983	1.1065	1.5006
1	512	9	1	2.2084	1.1834	1.1003	1.4974
1	64	6	1	2.0942	1.1421	1.0293	1.4218
8	512	1	0	2.2020	1.2851	1.1884	1.5578

*C. **result***

This table shows the best result we got and the parameter for it, it use PAp branch predictor:

TABLE V
BEST PREDICTION RESULT

L1	L2	hit	XOR	Anagram	go	gcc	AVG
8	512	1	0	2.2020	1.2851	1.1884	1.5578