# Branch Prediction

## ECE/CS 570 – High Performance Computer Architecture

Yu-Wen Tseng
Oregon State of University
*tsengyuw@oregonstate.edu*

Xinwei Li
Oregon State of University
*lixinwe@oregonstate.edu*

Chenyan Zhang
Oregon State of University
*zhangch5@oregonstate.edu*

***Abstract*—To observe the efficiency on the performance of pipelined and superscalar processors, the branch prediction has become an interesting area in computer architecture. There are lots of different ways, which have been proposed to speculate the path of an instruction stream after a branch. This paper presents the evaluation of the performance compared with different ideas using an existing simulation tool.**

***Keywords—branch prediction, simplecalar, benchmarks***

### I. INTRODUCTION

Branch prediction is used to resolve a branch hazard by predicting which path will be taken. To let a processor speculatively fetch across basic-block boundaries, branch prediction can be thought of as a sophisticated form of pre-fetch or a limited form of data prediction that tries to predict the result of branch instructions.

The purpose of this project is to understand branch prediction and explore its design space. Computer architects in industry use software-based micro architectural 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 associatively of caches, the size of RoB, the number of functional units etc. Besides, the branch prediction schemes chosen for this comparison are statically taken/not-taken, bimodal, combination, two-level adaptive (TLA), and gshare branch prediction schemes.

### II. TASK AND REQUIREMENTS

We will use “sim-outorder” to run different benchmarks. The guidance of the SimpleScalar can be found at linkhttp://www.simplescalar.com/docs/users_guide_v2.pdf.

Make sure that the simulator is configured as Alpha Simulator. Use “sim-outorder” to run benchmarks. We will need to modify config/default.cfg file to simulate different configurations.

### III. BASIC BRANCHMARK PREDICTION

#### A. *Benchmark Requirements*

- The tasks which will be listed below all need to run 3 benchmarks included in the downloaded benchmark folder: GCC, ANAGRAM (input data: anagram in), GO (input config: 50 92stone9.in)
- The instruction of running benchmarks are included in the benchmark folder.
- Use Alpha Benchmark

#### B. *The Processes of running data of benchmarks*

- Run the three benchmarks: replace <target> with the target of the machine simulated, either "alpha" for Alpha OSF, or "pisa-big" for PISA big endian
- Modify the branch prediction type, as Figure.4 shown, to different five benchmark type, which are Taken, Not Taken, Bimod, 2 level and Combined.
- Input the data by those instruction then comes the results:
  - GCC:
    to run: sim-safe cc1.<target> -O 1stmt.i
    to test output: diff 1stmt.s 1stmt.s.ref
  - ANAGRAM:
    to run: sim-safe anagram.<target> words < anagram.in > OUT
    to test output: diff OUT anagram.out
  - GO:
    to run: sim-safe go.<target> 50 9 2stone9.in > OUT
    to test output: diff OUT go.out


Fig. 1. Modify branch prediction type

### *C. Address-prediction rate for different branch predictors*

Execute Benchmarks and fill the Table I with address-prediction rate and plot a histogram for all the three benchmarks.

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. 2. Address-prediction rate

**Conclusion:**

1) The performance of Bimod predictor is better than 2-level predictor.
2) The taken and no-taken predictions rate are the same.

**Does your conclusion agree with your intuition?**

As for first conclusion, we thought that the Bimod predictor will not have performance better, and the 2-level predictor will more excellent.

### *D. IPC for different branch predictors*

Execute Benchmarks and fill the Table II with IPC and plot a histogram for all the three benchmarks.

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 |

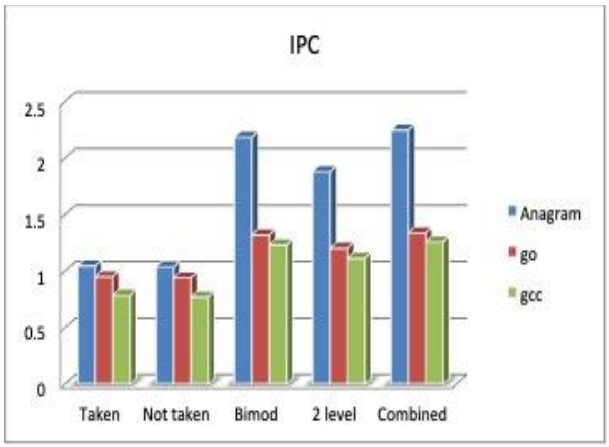


Fig. 3. IPC

### *E. Address prediction rate for different cache sizes*

```
# default sim-outorder configuration
#

# random number generator seed (0 for timer seed)
-seed                            1

# instruction fetch queue size (in insts)
-fetch:ifqsize                   4

# extra branch mis-prediction latency
-fetch:mplat                     3

# branch predictor type {nottaken|taken|perfect|bimod|2lev}
-bpred                        2lev

# bimodal predictor BTB size
-bpred:bimod                  2048

# 2-level predictor config (<l1size> <l2size> <hist_size>)
-bpred:2lev            8 8192 5 0

# instruction decode B/W (insts/cycle)
-decode:width                    4

# instruction issue B/W (insts/cycle)
-issue:width                     4
```

Fig. 4. Modify bimodal predictor table entry numbers

Use bimodal branch predictor, change the bimodal predictor table entry numbers, execute Benchmarks, and fill the Table III with address-prediction rate and plot a histogram for all the three benchmarks.

TABLE III. ADDRESS-PREDICTION RATE

| Benchmark | 256 | 512 | 1024 | 2048 | 4096 |
|---|---|---|---|---|---|
| **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. 5. Address-prediction rate

## IV. HOW COMBINED BRANCH PREDICOTR WORKS

In our group, three different branch predictors are used to be the combined branch predictor and analysis their results by a majority vote. To tracking the behavior of any branch, the multiple table entries are used by predictors. It will enlarge the possibility of two branches mapping to the same entry when entries are multiplied. However, the accuracy of the prediction will be influenced by those branches.

Moreover, because of several predictors, it will become more comparative and obvious results to have different aliasing patterns for each predictor. Therefore, apparently, it will have at least one predictor that has no aliasing [2].

The definition of combine branch predictors are:

- GAg: Global Adaptive Branch Prediction using one global pattern history table.
- GAp: Global Adaptive Branch Prediction using per-address pattern history tables.
- PAg: Per-address Adaptive Branch Prediction using one global pattern history table.
- PAp: Per-address Adaptive Branch Prediction using per-address pattern history tables.
- Gshare: Global History Register, Global History Table.

The basic method of combine branch predictors are:

- GAg : Hist Size W Pattern history is addressed with W bits
- GAp: 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. $Log_2N$ 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. $log_2N$ bits of PC taken.Branch history of each address, W bits.

Combines LSBs of PC and Branch History to address Pattern History

- Gshare: Table with Simple attempt at anti-aliasing

The Fig. 6 below is to show 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. 6. Relation between five predictors

## V. FIND THE BEST TWO -LEVEL BRANCH PREDICTOR

### A. Introduction

In SimpleScalar, the five 2-level branch predictor configurations will be simulated, including GAg, GAp, PAg, Pap and gshare. The task is to find the best 2-level branch predictor in terms of performance by changing parameters without exceeding storage constraints. We could only change four parameters of each 2-level branch predictor, including 11_size, 12_size, hist_size and XOR. The performance is defined as below:

Performance evaluation metric:

Average IPC = ($IPC_{GCC}$ + $IPC_{ANAGRAM}$ + $IPC_{GO}$) / 3

Furthermore, we need to make sure the storage consumption of the predictor (storage consumption = l1_size * hist_szie + l2_size * 2, unit: bits) does not exceed 4 KB. More information could be found in the user's guide.

### B. The predictor-specific arguments

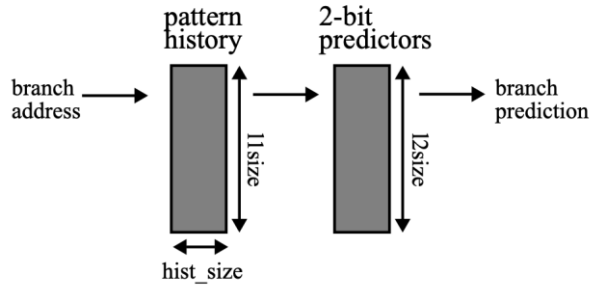


Fig. 7. 2-level adaptive predictor structure

-bpred:2lev <l1size> <l2size> <hist_size> <xor> specify the 2-level adaptive predictor. <l1size> specifies the number of entries in the first-level table, <l2size> specifies the number of entries in the second-level table, <hist_size> specifies the history width, and <xor> allows you to xor the history and the address in the second level of the predictor.

This organization is depicted in Fig. 7. In Fig. 8 we show how these parameters correspond to modern prediction schemes. The default settings for the four parameters are 1, 1024, 8, and 0, respectively.

| predictor | l1_size | hist_size | l2_size | xor |
|---|---|---|---|---|
| GAg | 1 | W | $2^W$ | 0 |
| GAp | 1 | W | $>2^W$ | 0 |
| PAg | N | W | $2^W$ | 0 |
| PAp | N | W | $2^{N+W}$ | 0 |
| gshare | 1 | W | $2^W$ | 1 |

Fig. 8. Branch predictor parameters

### C. Testing different combination of parameters

These five 2-level branch predictors have been tested to compare which type of performance is better than others, the data of each branch predictor are show in Table IV to Table VIII. As result shown, it is obvious that the performance of GAg and GAp are worse than others, and PAp has the better performance among other three.

TABLE IV. IPC FOR GAG

| GAg | L1 | L2 | Hist | XOR | Anagram | go | gcc | AVG |
|---|---|---|---|---|---|---|---|---|
| GAg (1) | 1 | 32 | 5 | 0 | 1.5964 | 1.12886 | 1.037 | 1.254 |
| GAg (2) | 1 | 512 | 9 | 0 | 1.6270 | 1.16222 | 1.09 45 | 1.295 |
| GAg (3) | 1 | 1024 | 10 | 0 | 1.6368 | 1.17555 | 1.11 05 | 1.308 |
| GAg (4) | 1 | 8192 | 13 | 0 | 1.6373 | 1.21335 | 1.14 12 | 1.331 |

TABLE V. IPC FOR GAP

| GAp | L1 | L2 | Hist | XOR | Anagram | go | gcc | AVG |
|---|---|---|---|---|---|---|---|---|
| **GAp (1)** | **1** | 64 | 5 | 0 | 1.7525 | 1.1406 | 1.0580 | 1.317 |
| **GAp (2)** | **1** | 1024 | 9 | 0 | 1.7863 | 1.1883 | 1.1439 | 1.373 |
| **GAp (3)** | **1** | 2048 | 10 | 0 | 1.7920 | 1.2041 | 1.1618 | 1.386 |
| **GAp (4)** | **1** | 8192 | 12 | 0 | 1.7957 | 1.2363 | 1.1879 | 1.407 |

TABLE VI. IPC FOR PAG

| PAg | L1 | L2 | Hist | XOR | Anagram | go | gcc | AVG |
|---|---|---|---|---|---|---|---|---|
| **PAg (1)** | 2 | 32 | 5 | 0 | 1.5760 | 1.1250 | 1.0386 | 1.247 |
| **PAg (2)** | 4 | 512 | 9 | 0 | 1.7436 | 1.1516 | 1.1019 | 1.332 |
| **PAg (3)** | 8 | 1024 | 10 | 0 | 2.0937 | 1.1714 | 1.1241 | 1.386 |
| **PAg (4)** | **16** | 8192 | 13 | 0 | 2.1538 | 1.1821 | 1.1695 | 1.463 |

TABLE VII. IPC FOR PAP

| PAp | L1 | L2 | Hist | XOR | Anagram | go | gcc | AVG |
|---|---|---|---|---|---|---|---|---|
| **PAp (1)** | 1 | 64 | 5 | 0 | 1.7525 | 1.1406 | 1.0580 | 1.317 |
| **PAp (2)** | 4 | 512 | 5 | 0 | 2.1991 | 1.2059 | 1.1507 | 1.332 |
| **PAp (3)** | 8 | 2048 | 3 | 0 | 2.2170 | 1.3075 | 1.2469 | 1.519 |
| **PAp (4)** | 8 | 4096 | 4 | 0 | 2.2151 | 1.3173 | 1.2603 | 1.600 |

TABLE VIII. IPC FOR GSHARE

| gshare | L1 | L2 | Hist | XOR | Anagram | go | gcc | AVG |
|---|---|---|---|---|---|---|---|---|
| **gshare (1)** | 1 | 32 | 5 | 1 | 2.0026 | 1.1350 | 1.0430 | 1.394 |
| **gshare (2)** | 1 | 512 | 9 | 1 | 2.2084 | 1.1833 | 1.14 | 1.511 |
| **gshare (3)** | 1 | 1024 | 10 | 1 | 2.2430 | 1.2026 | 1.1795 | 1.542 |
| **gshare (4)** | 1 | 8192 | 13 | 1 | 2.2565 | 1.2594 | 1.2624 | 1.593 |

### *D. Results*

According to compare 5 configurations, we observed that the PAp improved the best performance. Therefore, we chose 10 PAp data to find which one with different L1 size, L2 size, hist size and XOR has the best performance. As the Table IX shown.

TABLE IX. TESITING SEVERAL PAP

| PAp | L1 | L2 | Hist | XOR | Anagram | go | gcc | AVG |
|---|---|---|---|---|---|---|---|---|
| **PAp (1)** | 1 | 64 | 5 | 0 | 1.7525 | 1.1406 | 1.0580 | 1.317 |
| **PAp (2)** | 2 | 128 | 5 | 0 | 1.8492 | 1.1514 | 1.0896 | 1.363 |
| **PAp (3)** | 4 | 256 | 4 | 0 | 2.1932 | 1.1944 | 1.1271 | 1.505 |
| **PAp (4)** | 4 | 512 | 5 | 0 | 2.1991 | 1.2059 | 1.1507 | 1.332 |
| **PAp (5)** | 4 | 1024 | 6 | 0 | 2.2096 | 1.2165 | 1.1749 | 1.534 |
| **PAp (6)** | 8 | 2048 | 3 | 0 | 2.2170 | 1.3075 | 1.2469 | 1.519 |
| **PAp (7)** | 8 | 4096 | 4 | 0 | 2.2151 | 1.3173 | 1.2603 | 1.600 |
| **PAp (8)** | 4 | 4096 | 8 | 0 | 2.2105 | 1.2347 | 1.2164 | 1.554 |
| **PAp (9)** | 4 | 8192 | 9 | 0 | 2.2139 | 1.2449 | 1.2346 | 1.564 |
| **PAp (10)** | 8 | 8192 | 5 | 0 | 2.2151 | 1.3276 | 1.2737 | 1.605 |

As for above table, it comes the best result and the parameters as below:

TABLE X. BEST RESULTS

| L1 | L2 | Hist | XOR | Anagram | go | gcc | AVG |
|---|---|---|---|---|---|---|---|
| **8** | 8192 | 5 | 0 | 2.2151 | 1.3276 | 1.2737 | 1.605 |

## REFERENCES


[1] Yeh, Tse-Yu, and Yale N. Patt. "A comparison of dynamic branch predictors that use two levels of branch history." *Proceedings of the 20th annual international symposium on computer architecture*. 1993.

[2] J. V. Hansen, “Combining Predictors,” DAIMI Report Series, vol. 29, no. 550, Jan. 2000.

[3] 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.