Final project: Phase 3

Two-mode branch predictor

Group 5:
Yunfei Xu G21534523
Xiaohan Li G47313418
Haoran Deng G41396016

Instructor: Nahid Farhady GhalatyAuthors
Computer System Architecture, 6461.10
May 04, 2018
The George Washington University

***Abstract*—Branch prediction is a long-lasting topic to overcome the performance limitation of modern high-performance architecture. This report will focus on correlated branches prediction, propose a new two-mode (hybrid) branch predictor according to 2-level global branch predictor, compare it to the existed correlated predictor by using the simulator-SimpleScalar.**

Keywords: new hybrid branch predictor, global two-level branches predictor, SimpleScalar

I. INTRODUCTION

Branch prediction is a normally way to reduce overall branch misprediction and the number of instructions executed on the wrong path to save cost. The correct prediction of branch outcomes and targets is necessary to avoid pipeline bubbles [1]. Right now, several types of predict module have been proposed, such as static branch prediction and dynamic branch prediction.

If a program only contains simple statements, static branch prediction, such as one-level (bimodule) predictor can deal it, which use a Branch History Buffer (BHB) to find the specific entry according to the branch address and use a counter indicate whether the branch was recently taken.

While in complex program, branches are not isolated, the outcome of a branch may correlate to the previous outcomes of itself, using only static branch prediction to record branch address cannot get the predict result accuracy. Dynamic branch prediction uses outcomes of previous occurrences of branches to dynamically predict the outcome of the current branch. Modules created by Yeh and Patt [2], using global (and local) branch history to output the prediction, such as TAGE-based predictors [3] and the neural-inspired predictors [4]. These predictors allow the behavior of other branches to also update the predictor bits for a branch instruction and achieve slightly better overall prediction accuracy.

In this report, I will show that a branch result will not only be correlated to its previous results, but also correlated by the results of other branches. So that is the reason why we design a new predictor to handle it.

The outline of this report shows there: Section 2 and 3 introduces two kinds of existed two-level predictor to handle self-correlated branch prediction. And shows how it can work and why sometimes it cannot work.

Section 4 presents our proposed method: two-mode branch predictor, we talk about its architecture, prediction way, counter updating algorithm and its improvement compared with former predictors.

The rest sections introduce the simulator we use and shows the experimental results we get by using simulator and benchmarks.

II. TWO-LEVEL BRANCH PREDICTORS

A branch's outcome can be correlated with branch's previous outcomes. To record these outcomes in history, two-level adaptive branch predictors were created.

A. Global two-level adaptive branch predictor

The outcome of branches (of any address) is recorded in the global shift register. If a branch is taken, 1 is inserted into the shift register and the shift register is shifted; if untaken - 0 is inserted into the shift register and the shift register is shifted. Branch predictions are made by checking the pattern history bits in the pattern table entry indexed by the content of the history register for the branch that is being predicted. [2]

The pattern history bits are counters S_c. Using the algorithm λ (S_c) to get the prediction of branch. When we get the actual result of this branch, we can calculate the S_{c+1} according to the result and S_c, then we update the S_c into S_{c+1} and update the BHR. [1]

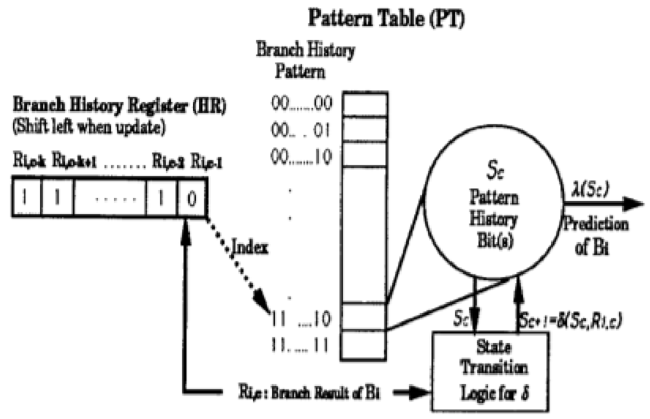


Figure 1. Global Two-level adaptive branch predictor.

B. MCFarling's Two-Level Prediction

Gshare algorithm is the improvement of two-level adaptive branch predictor. It hashes global BHR and branch address together. Using the XOR of BHR and branch address to index the pattern history table(PHT).

GShare add more context information and better utilization of PHT.

C. Local two-level adaptive branch predictor

The first table is the local branch history table. It is indexed by PC address and records the taken/not-taken history of the n most recent executions of the branch, which record recent outcomes for different branch in different entry to avoid non-useful branches' noise.

The other table is the pattern history table, like the table in global branch predictor. However, its index is generated from the branch history in the first table. To predict a branch, the branch history is looked up, and that history is then used to look up a bimodal counter which makes a prediction.

Researches finished by André Seznec [5] and Y. Ishii [6] focus on local history component to create a new predictor to predict those correlated branches.

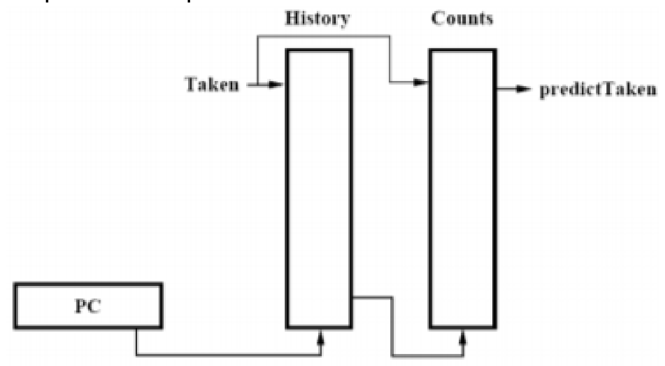
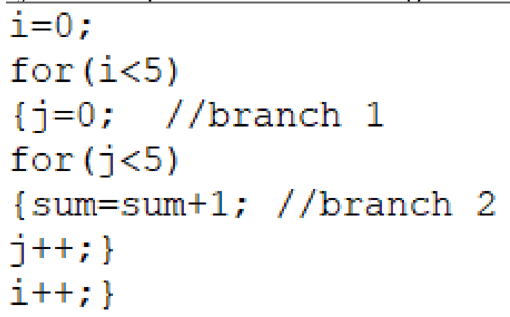


Figure 2. Local branch predictor.

D. Problem can solve

Such predictor can predict branch result using local transfer history, which contains the previous results of branch itself. So, such predictor works successful especially in for loop iteration shows in Figure 3.



```
i=0;
for(i<5)
{j=0;  //branch 1
for(j<5)
{sum=sum+1; //branch 2
j++;}
i++;}
```

Figure 3. Branch correlated to itself.

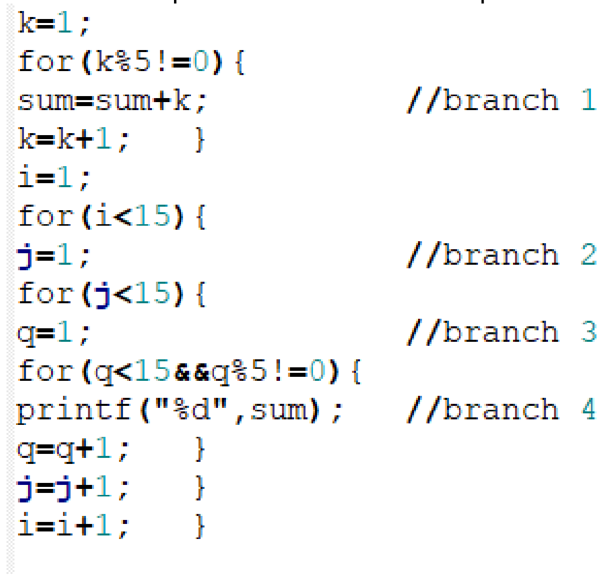
Both branch 1 and branch 2 will have 5 taken and 1 not taken, two-level predictor, especially local two-level predictor can prepare different history table entry to store history results of a specific branch, and make prediction using that result.

If a program contains many loop iteration, two-level adaptive branch predictor can make prediction accuracy in a great degree.

E. Problem cannot solve

But we must know that a branch's result maybe not only correlated by its previous results, but also correlated by the other branches in the same iteration. Figure 4 shows an example.

If we know the result of branch 2, then we can predict branch 3 according to 2, which can be solved by two-level predictor. But to predict branch 4, only branch 2's result is not enough, we still need result of branch 1 to make prediction. In that case, only using two-level branch predictor cannot solve the problem.



```
k=1;
for(k%5!=0){
sum=sum+k;          //branch 1
k=k+1;    }
i=1;
for(i<15){
j=1;                //branch 2
for(j<15){
q=1;                //branch 3
for(q<15&&q%5!=0){
printf("%d",sum);   //branch 4
q=q+1;    }
j=j+1;    }
i=i+1;    }
```

Figure 4. Branches correlated to other branches.

III. PROPOSED METHOD: TWO-MODE PREDICTOR

The predictor we design to solve the problem I proposed is to use a new predictor, combines with the GShare type two-level adaptive branch predictor, Branch History Pattern indexed by the XOR result of branch address and branch history register, to form a two-mode (hybrid) predictor. I will also classify which branches should be used in the two-mode predictor, and how to divide them, to improve the accuracy of prediction.

A. New predictor we design

We set a Correlated Branch Register, indexed (hashed) by branch address, to represent all correlated branches. Consider that maybe in the same program it will both contain A and !A, Only use 1 bit cannot represent 3 condition: X is correlated to B, X is correlated to !B and X is not correlated to branch B (and !B), so we use 2 bits to represent each branch, B and !B are represented in different bit.

It will record 0 and 1 to represent whether a branch is correlate to the branch we predict now, and Correlated Branch Register will index the entries in the Pattern Table to get the counter. Clearly, this predictor is a static predictor, not dynamic, we don't need to store history information. The new part shown in Figure 5.

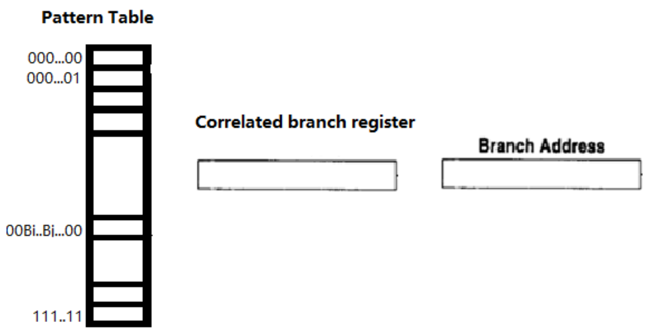


Figure 5. New part of predictor.

For example, I set B_1 to B_4 are basic branches, and the predictions of B_i (i>4) are correlated by some of these 4 branches. The correlated branch register will be 8-bits long in common, so we can represent the register as $R_1R_1' R_2R_2' R_3R_3' R_4R_4'$, If we set B_5's outcomes are correlated by B_2 and $!B_4$, then the bits of Correlated Branch Register for B_2 and $!B_4$ to 1 and others to 0. So, the hash result of B_5 branch address will be 00100001, which will index an entry in the Pattern Table to get a 2-bit history bit.

In each program, we need to confirm whether B and !B will both be used, if we find $!B_1$ and B_3 will never exist, then the correlated branch register can be changed to $R_1R_2R_2'R_3'R_4R_4'$, a 6-bit register, to decrease pattern table size. That will not influence the prediction because those entries will never be used.

So, for the prediction of B_1 to B_4, we still use GShare 2-level branch predictor to predict. Hash result of address B_1 to B_4 is 0. Only those branches whose address hash result is not 0 (means are correlated by B_1 to B_4) will use both parts to get a combined result.

B. Whole architecture of two-mode predictor

Figure 6 shows the whole architecture of the new predictor, divided by 5 parts: Branch History Register (BHR), Pattern History Table (PHT), Pattern Table, Correlated Branch Register and state transition logic.

Branch history register will store the most recent branch outcomes by shifting left one bits per prediction; Using the XOR result of branch address and BHR, to index the PHT, each entry will store a 2-bit saturating counters (pattern history bits) for the prediction.

Correlated Branch Register will be used only for correlated branches, that will have decided by branch address. Set those correlate branches' bits to 1 and others to 0 and using a new hash function and branch address to correspond one by one; Pattern Table will store the latest result for specific correlate branches combination by using 2-bit saturating counters (pattern history bits).

Such correlated branches will get 2 2-bit pattern history bits from PHT and Pattern Table. Combining them as a 4-bit counter (PHT 2bit left, PT 2 bit right) to do the prediction.

After prediction by using either 2-bit or 4-bit counter, the branch gets its result, the state transition logic will calculate the new pattern history bit according them and update it to the Table (or to the two Tables separately).

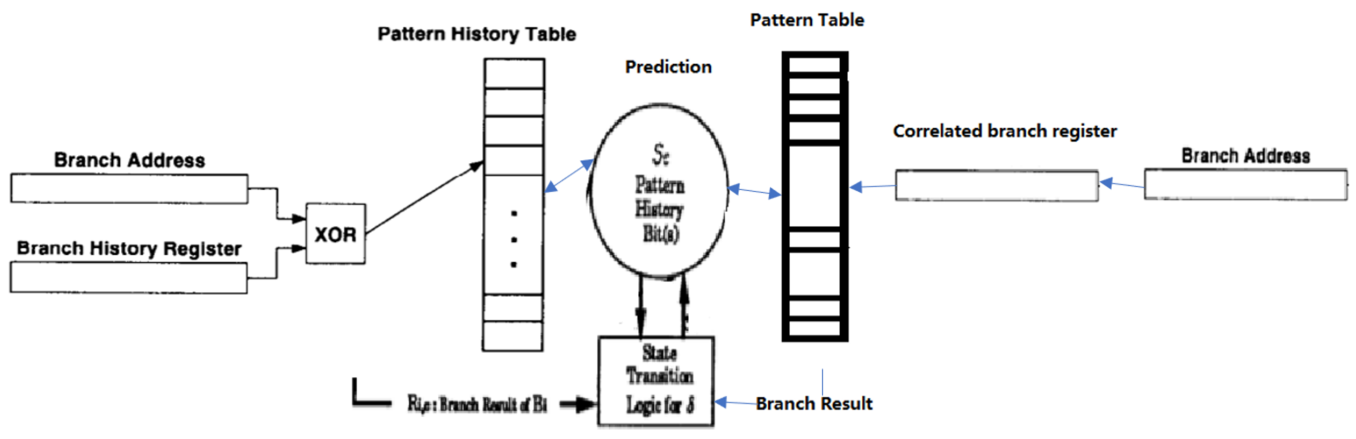


Figure 6. Two-mode branch predictor

We use the program in Figure 4 to explain the whole step of my predictor:

Branch 1 and 2 are basic branches. The prediction of them will be finished only by GShare two-level adaptive predictor and result will be stored in PHT.

Branch 3 and Branch 4 are correlated by Branch 1 and 2, so we use two-mode predictor to predict them. Because there are only 2 branches are correlated, and $!B_1$ and $!B_2$ will never be used, so we can set the Correlated Branch Register to 2-bit size to represent B_1 and B_2.

Branch 3 is only decided by Branch 2, so the Correlated Branch Register will be 01. The Branch 4 not only need Branch 2 but also need Branch 1. So, the Correlated Branch Register will be 11. We find the entry indexed in both PHT and Pattern Table and predict an outcome using the 4-bit counter.

C. Pattern history bit updating in two-mode predictor

For the global two-level history table, the pattern history bit's update algorithm will not be changed. But when we use two-mode predictor and get a 4-bit history bit, the algorithm need to be created.

Figure 7 shows the four-bit predictor state transition diagram. The first two bits represent the history bit stored in PHT (global 2-level predictor), and the last two bits represent the history bit stored in Pattern Table. (new mode predictor)

If the counter in PHT is 10, and the counter in PT is 11, then the 4-bit history bit is 1011, we should predict taken at first, if the prediction is correct, then the counter will become 1100, and the new counter in PHT will be 11 and the new counter in PT will be 00.

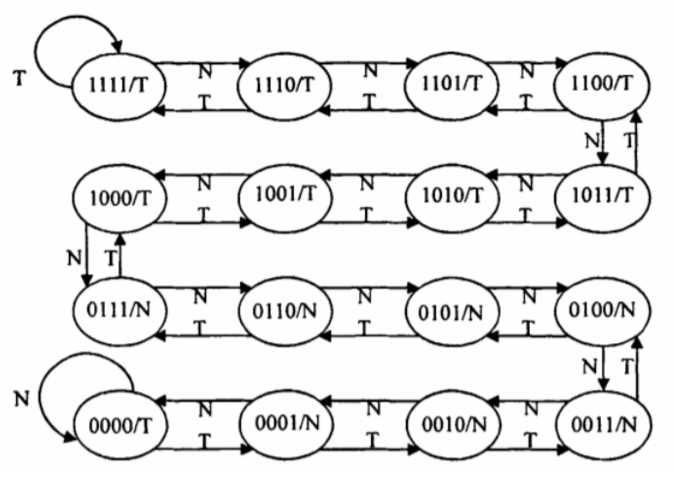


Figure 7. Four-bit Predictor State Transition Diagram.

D. Difference compared to previous predictors

New predictor is very similar to a kind of hybrid predictor, which combining global two-level adaptive predictor and two-bit bimodal predictor.

1) new-mode and bimodal predictor:

New designed part of the predictor is very similar to the 2-bit bimodal predictor. The difference is the Pattern Table's size and entries are decided and indexed by Correlated Branch Register, not PC address. Correlated Branch Register brings flexibility to the predictor. If we must use at least 10 bits in PC address to distinguish all branches, then the Pattern Table's size is equal to 2^{10} large no matter how many entries will be used.

In this predictor, if branches are correlated by 3 previous branches $B_1B_2B_3$, the Correlated Branch Register will be 6-bit long (both B and !B) and the Pattern Table will be 2^6 large. If we have more than 5 correlated branches $B_1B_2B_3B_4B_5$, the size of Pattern Table will larger than 2^{10}.

2) two-mode and hybrid predictor:

Except the new designed part, we also have some difference between the hybrid predictor I introduced before and the two-mode predictor we designed.

At first, the original predictor only uses global history register to index the Pattern History Table, but our two-mode predictor uses GShare algorithm to add more context information and better utilization of PHT. All assumption of this report is some branches are correlated to others. And different branch represents different branch address, using GShare algorithm to XOR branch address and global history register at first will increase the relevant to branches themselves.

Second, when the original predictor using two predictors at the same time, a special Branch Predictor Selection Table, or meta-predictor, must be used to choose which predictor to use for a given situation. This meta-predictor is like the two-bit predictor discussed above. It uses a two-bit counter to keep track of which predictor is more accurate for a specific branch address. In other words, only one predictor will be used.

In our two-mode predictor, we only use global two-level adaptive predictor when the hash result of PC address into Correlated Branch Register is 0. But in other case, when the Correlated Branch Register have bits that are 1, we need to use two predictors at same time to get a 4-bit history pattern. So, the meta-predictor will be cancelled, each time we need to calculate the new counter to finish prediction. It is also clear that the new-designed part of the predictor will never be used independently, either be the last two bits of the history bit, or not be used.

IV. SIMPLESCALAR

The SimpleScalar tool set provides an infrastructure for simulation and architectural modeling. It can model a variety of platforms ranging from simple unpipelined processors to detailed dynamically scheduled microarchitectures with multiple-level memory hierarchies. SimpleScalar simulators reproduce computing device operations by executing all program instructions using an interpreter. The tool set's instruction interpreters also support several popular instructions sets, including Alpha, PPC, x86, and ARM. [5]

V. EXPERIMENTAL STEPS AND RESULT

To show the result of our new mode, I use 4 kinds of benchmarks downloaded in SimpleScalar LCC: gcc, anagram, compress, go; using 5 branch prediction modes: nottaken, taken, bimod(2048), 2lev(1 1024 8 0), comb(2lev: 1 1024 8 0, bimod: 2048, meta: 1024), all in default mode, to compare with the two-mode branch prediction(2lev: 1 1024 10 1, bimod: 2048), in default mode.

I using alpha mode and excute cc1.alpha, anagram.alpha, compress95.alpha, go.alpha for all mode above, recording branch address-prediction rate and branch direction-prediction rate, the result shown in Figure 10-13.

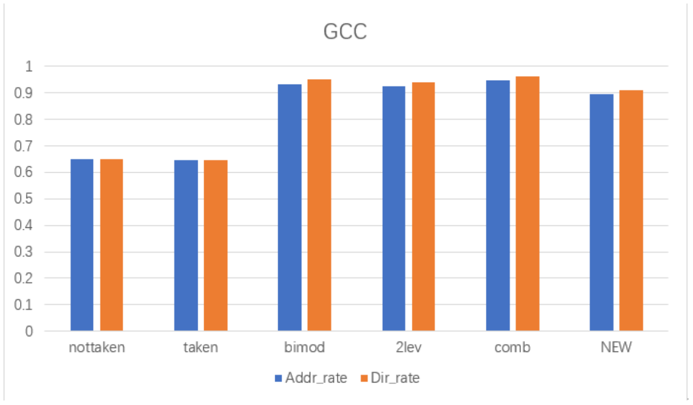


Figure 8. GCC

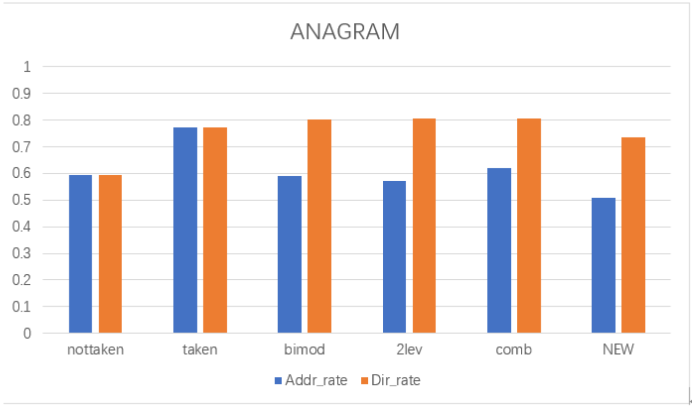


Figure 9. Anagram

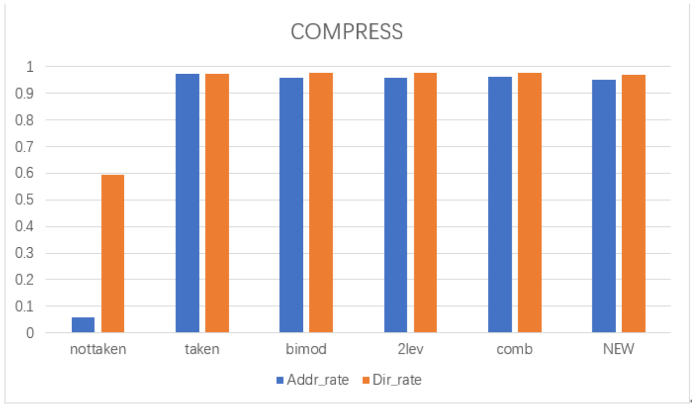


Figure 10. Compress

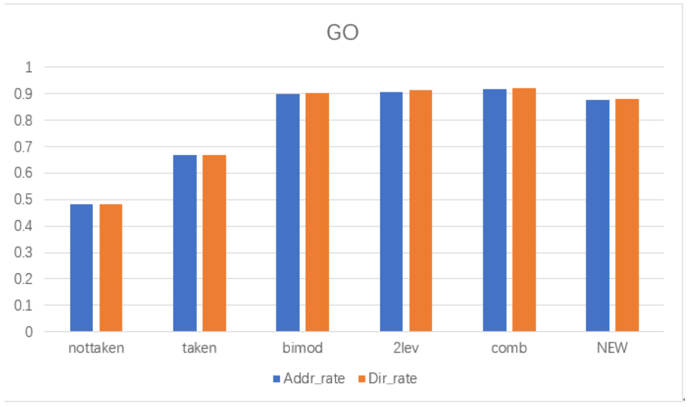


Figure 11. Go

VI. CONCLUSION

From the results shows before, I found the accuracy of our two-mode predictor is not as high as some original modes. The possible reason we guess is the 4-bit counter make the prediction more difficult.

For a 4-bit counter which value is 1111, we need 8 times continuous misprediction to let the counter be 0111 and predict not taken. While a 2-bit counter which value is 11 only need 2 times misprediction that can correct its prediction.

Although our new design predictor doesn't increase the predict accuracy. But I believe that using correlated branches to index counter will useful and we will keep finding the better solution.

REFERENCES

[1] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt. An analysis of correlation and predictability: What makes twolevel branch predictors work. In Proceedings of the 25th Annual International Symposium on Computer Architecture, pages 52 – 61, 1998.

[2] T.-Y. Yeh and Y. Patt, "Two-level adaptive branch prediction," in Proceedings of the 24th International Symposium on Microarchitecture, Nov. 1991.

[3] A. Seznec, "Tage-sc-l branch predictors," in Proceedings of the 4th Championship on Branch Prediction, http://www.jilp.org/cbp2014/, 2014.

[4] R. S. Amant, D. A. Jiménez, and D. Burger, "Low-power, high-performance analog neural branch prediction," in MICRO,pp. 447–458, 2008.

[5] A. Seznec, J. S. Miguel and J. Albericio, "The inner most loop iteration counter: A new dimension in branch history," 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Waikiki, HI, 2015, pp. 347-357.

[6] Y. Ishii, K. Kuroyanagi, T. Sawada, M. Inaba, and K. Hiraki, " Revisiting local history for improving fused two-level branch predictor," in Proceedings of the 3rd Championship on Branch Prediction, http://www.jilp.org/jwac-2/, 2011.

[7] Y. Ishii. Global-local combined branch history: The alternative way to improve TAGE branch predictor. In JWAC-4: Championship Branch Prediction. JILP, June 2014.

[8] T. Austin, E. Larson and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," in Computer, vol. 35, no. 2, pp. 59-67, Feb 2002.

[9] KleinOsowski, A. J., and David J. Lilja. "MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research." IEEE Computer Architecture Letters 1.1 (2002): 7-7.

[10] Jeppsson, Ulf, et al. "Benchmark simulation model no 2: general protocol and exploratory case studies." Water Science and Technology 56.8 (2007): 67-78. Electronic Publication: Digital Object Identifiers (DOIs):

Appendix - Installation Guidance

1. SimpleScalar installation.

All using files are in the Document zip.
simplesim-3v0e.tgz, simpletools-2v0.tgz, simpleutils-2v0.tgz are stored in build folder.
Using scrpt file: install_simple_scalar.sh to install software.
Instruction: *./install_simple_scalar.sh*.

2. Test original mode using benchmarks.

All benchmarks are contained in benchmarks folder.

A. Compile SimpleScalar Instruction
make config-alpha
make

B. Benchmarks instruction

4 benchmarks using different instructions:
GCC: *cc1.<target> -O 1stmt.i*
ANAGRAM: *anagram.<target> words < anagram.in > OUT*
COMPRESS: *compress95.<target> < compress95.in > OUT*
GO: *go.<target> 50 9 2stone9.in > OUT*

C. Branch predictor specification

specifying the branch predictor type: -bpred <type>
Taken: *./sim-bpred -bpred taken*
Not Taken:*./sim-bpred -bpred nottaken*
Bimodal predictor*: ./sim-bpred -bpred:bimod <size>*
2-level adaptive predictor: *./sim-bpred -bpred:2lev <l1size> <l2size> <hist_size>*
Combined predictor (bimodal and 2-level adaptive): *./sim-bpred -bpred comb <size>*

So the final instruction will be something like that:
./sim-bpred -bpred <type> cc1.<target> -O 1stmt.i

3. New mode

D. Recompile the simplescalar

Replace the bpred.c, bpred.h, sim-bpred.c and sim-outorder.c file in simplesim-3.0 to the files stored in Modified code folder.

Clean instruction: make clean
Compile: make config-alpha
make

E. Testing

The new mode will be executed by comb mode.
./sim-bpred -bpred comb

The example instruciton:
./sim-bpred -bpred comb compress95.<target> < compress95.in > OUT