ECE1718 Project Report

Hao Jun Liu

I. Introduction

Modern architectures exploit instruction level parallelism through mechanism such as superscalar and out of order execution. Statistics have shown that there is on average one control flow instruction in every five instructions. Thus, in order for the instruction scheduler to find as many instructions to execute, the fetching engine need to fetch enough instructions to feed the function units without knowing the exact control flow of the program. In other word, the fetch engine need to fetch instructions speculatively without resolving the branches. In the case of 4-way out-of-order machine with a window of 128 instructions, the predictor need to correctly speculate 128/5≈26 control flow instructions, on average one in each cycle, in order to fill up the instruction window. Assume the prediction hit rate for a set of programs is 0.99 which is a very high prediction rate, the chance that the fetch engine can fill up the instruction window is only 0.99^{26}=0.77. Therefore, branch predictions are necessary for an superscalar OOO machine to have enough instructions for scheduling. I implement and simulate several state of the art branch predictors including *2BcgSkew*, *O-GHEL*, *TAGE* and *YAGS*. It is expected that TAGE would be the best one according to previous work[1]. However it is also very hard to tune since there are so many freedom in a TAGE predictor especially the update policy. I find that TAGE is very sensitive to predictor update policy among the four predictors. The best one in terms of average misprediction per thousand instructions among the four predictors is O-GHEL. The evaluation of those predictor shows they have an average misprediction per thousands instructions(MPKI) between 5.55 and 7.57 for the spec2k trace provided when the simulator executes 30M instructions. The best performance is 4.37 for *O-GEHL* when run all benchmarks to complete.

II. Related Work

All the predictors implemented and simulated in this project are based on previous published works with some modification. The *2BcgSkew* predictor is based on the work of [2] and [3]. The *O-GHEL* predictor is based on the work of [4] and [5]. The *TAGE* predictor is based on the work of [1]. The *YAGS* predictor is based on the work of [6] and [7]. According to the above works, *TAGE* would be the one with best performance among four different branch predictors. My work are all based on the above works with some modification or enhancement which will be described in the next section.

III. Methodology

This section describes the detail predictor configurations for *2BcgSkew, O-GHEL*, *TAGE* and *YAGS*. All of the them have a storage budget of 64KB. Global history storage is not counted in the 64KB budget. There are minor differences between the predictor described in previous work and my implementation. This section gives a brief description of the predictor implemented and focuses on the difference between.

Predictor	Number of Entry	Global History Length	Storage (Bits)
BIM	32768	0	65536
G0	98304	6	196608
G1	98304	18	196608
Meta	32768	0	65536
		Total(KB)	64

Fig. 1. 2BcgSkew Configuration

A. 2BcgSkew

2BcgSkew is a hybrid predictor including a gSkew predictor which has three predictor banks and a bimodal predictor. The bimodal predictor is shared with gSkew predictor. There is a meta predictor that selects which predictor to use. Prediction tables are indexed using PC or a hashing function taking the PC and global branch history as input.

1) Hardware Configuration: Fig. 1 provides the hardware configuration for *2BcgSkew* used in the simulation.

2) Update Policy:

- On a bad prediction, the three banks of the *e-gskew* predictor are updated.
- On a correct prediction, only the banks participating to the correct prediction are updated.
- The meta predictor is updated to the direction of the correct predictor on each prediction.

B. O-GEHL

O-GEHL is a an optimized version of geometric history length predictor. The design used in this project has 9 components. The prediction is made by adding counter values in all components which use different length global history in a geometric pattern. It is build with a dynamic threshold fitting mechanism described in [5].

1) Hardware Configuration: Fig. 2 on the next page provides the hardware configuration for *O-GEHL* used in the simulation.

2) Update Policy: The *O-GEHL* predictor update policy is derived from the perceptron predictor update policy. The *O-GEHL* predictor is updated on the following scenarios

- On a bad prediction, all tables are updated to the direction of final branch outcome.
- If the absolute value of the computed sum S is smaller than a threshold θ, all tables are updated to the direction of final branch outcome regardless of the branch prediction correctness.

Table	T0	T1	T2	T3	T4	T5	T6	T7	T8
History	0	3	5	8	12	19	31	49	69
Counter	5	6	6	5	5	5	5	5	5
Entries	8192	16384	16384	16384	8192	8192	8192	8192	8192
								Total(KB)	64

Fig. 2. O-GEHL Configuration

- The threshold θ is updated if a TC counter is saturated. This is described in [5]. The rationale behind is different programs have different preference of the threshold θ.
- The TC is incremented when prediction is different from branch result and is decremented when prediction is correct but predictor sum S is less than threshold θ.

C. TAGE

TAGE is a tagged geometric history length predictor. The design used in this project has 8 components. The prediction is made by an tag matched entry in the table using the longest possible global history if the useful counter is not zero. If the useful counter is zero, the prediction is made by the component that use the second longest possible global history on cases where it gives higher prediction accuracy.

1) Hardware Configuration: Fig. 3 on the following page provides the hardware configuration for *TAGE* used in the simulation.

2) Update Policy:

- On a bad prediction, provider component is updated, a new entry using longer history is also allocated if possible.
- On a correct prediction, provider component's counter is updated, provider component's u counter is incremented.
- On a bad prediction, provider component's u counter is decremented.

D. YAGS

YAGS is based on bi-Mode predictor [7]. It uses a meta predictor with two gShare-like predictor with tag.

1) Hardware Configuration: Fig. 4 on the next page provides the hardware configuration for *YAGS* predictor used in the simulation.

2) Update Policy:

- Only the direction PHT chosen is updated.
- Choice PHT is not updated if it give a prediction contradicting the branch outcome and direction PHT chosen gives the correct prediction.

E. Baseline

Baseline predictor is a level two predictor with one meta predictor, one gShare predictor and one bimodal predictor. Each of the predictor has 64K entries and the ghsare predictor uses a global history length of 10. It is implemented in the simplescalar simulator suite.

IV. Evaluation

This section presents experimental result showing the *O-GEHL* predictor implemented achieves better performance than the baseline and other implemented predictors. It also shows that which predictor is better in terms of ideal performance and realistic performance.

A. Experimental Methodology

The experimental data presented in this report were collected using SPEC2K benchmark traces. The traces are compiled to executed on simplescalar PISA architecture which is very similarly to MIPS like architecture. The metric used is misprediction per thousand instructions. Only direction prediction for conditional branch is considered. That is, only conditional branch are predicted and the prediction is considered correct if the direction is correct. In order to get both a realistic performance and an ideal performance, the traces are first run with 30M instructions to simulate OS context switch and then run to complete. Moreover, for the realistic run, predictor initial state are randomized to simulate the processor state after a context switch. The realistic results are going to be used for performance evaluation. The ideal results is going to be used for the branch predictor competition.

B. Results

This section includes simulation result and analysis of the result of the predictors simulated. The last part of it is the result for ideal simulators which run the entire SPEC2K traces to complete.

1) 2BcgSkew: Fig. 5 on the following page shows the misprediction rate for the baseline predictor and *2BcgSkew*. Average MPKI for *2BcgSkew* is 6.99 while average MPKI for baseline is 7.15. We can see that the performance of *2BcgSkew* is a little bit better compared with the baseline predictor. This difference is insignificant in determining which predictor is better. By varying global history length used in different predictor banks and using different hashing function, I get the conclusion that the difference between the two is largely determined by minor difference in hashing function used in indexing the prediction table. The rational behind both of the predictors are almost the same. Give they have same storage budget, it does not surprise that they have very similar performance indeed.

2) O-GEHL: Fig. 6 on the next page shows the misprediction rate for the baseline predictor and *O-GEHL*. Average MPKI for *O-GEHL* is 5.55 while average MPKI for baseline is 7.15. We can see that the performance of *O-GEHL* is significantly better than the baseline predictor in terms of MPKI. The configuration of the *O-GEHL* is determined through a iterative

Table	T0	T1	T2	T3	T4	T5	T6	T7
History	0	5	9	15	25	44	76	130
Counter	2	2	2	2	2	2	2	2
Tag Width	0	9	9	9	10	11	12	12
u Witdh	0	2	2	2	2	2	2	2
Entries	32768	2048	4096	8192	8192	4096	4096	2048
Storage	65536	26624	53248	106496	114688	61440	65536	32768
							Total(KB)	64

Fig. 3. TAGE Configuration

Predictor	Number of Entry	Global History Length	Storage (Bits)
NT Cache	32768	12	196608
PHT Choice	65536	0	131072
T Cache	32768	12	196608
		Total(KB)	64

Fig. 4. YAGS Configuration

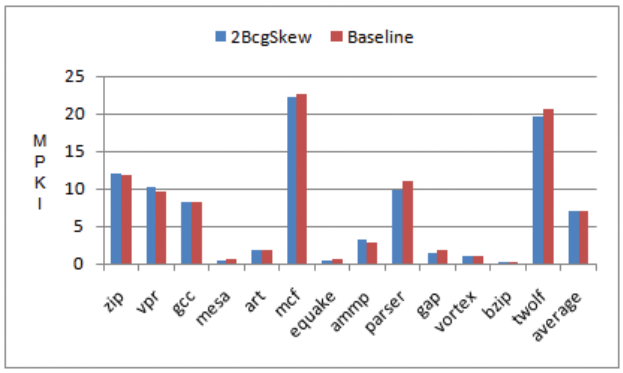

Fig. 5. 2BcgSkew Performance

approach. It has been found that O-GEHL is very sensitive to the maximum history length in situations where no dynamic history length mechanism is used. I get the conclusion that the performance of O-GEHL is significantly better than the baseline predictor. I can also reproduce some of the result in [4] and [5]. For example, without dynamic threshold fitting, the predictor suffers poor performance for some benchmarks.

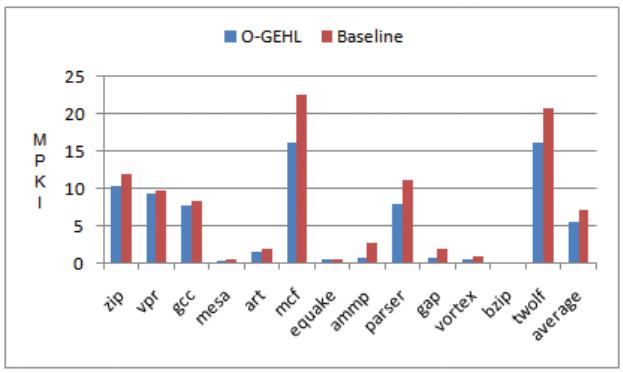

Fig. 6. O-GEHL Performance

3) TAGE: Fig. 7 shows the misprediction rate for the baseline predictor and *TAGE*. Average MPKI for *TAGE* is 6.12 while average MPKI for baseline is 7.15. We can see that the performance of *TAGE* is significantly better than the baseline predictor in terms of MPKI. The configuration of the *TAGE* is determined through an iterative approach. It has been found that TAGE is very sensitive to the update policy used. However, I am not able to reproduce the result in [1] where the *TAGE* is better than *O-GEHL* give that they have same amount of storage. However, due to the short time frame of this project, I cannot say that the *TAGE* predictor is very well optimized. I believe there are still improvements can be done on this predictor.

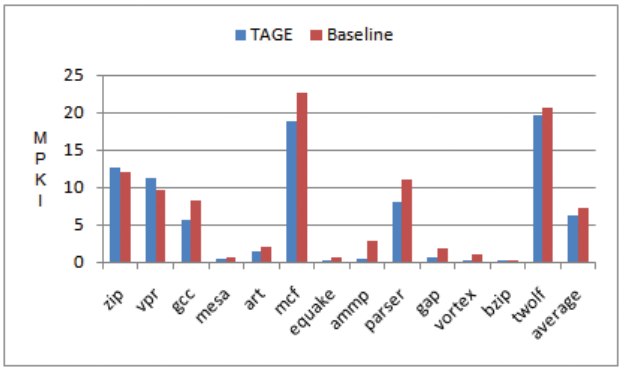

Fig. 7. TAGE Performance

4) YAGS: Fig. 8 on the next page shows the misprediction rate for the basline predictor and *YAGS*. Average MPKI for *YAGS* is 7.57 while average MPKI for baseline 7.15. We can see that the performance of *YAGS* is little bit worse than the baseline predictor in terms of MPKI. The configuration is determine through an iterative approach. It has been found that although predictor with tag can solve aliasing problem, it also uses precious storage resources thus the performance is not better compared with baseline and *2BcgSkew*. I reach a conclusion that *YAGS* has almost no advantage compared with *2BcgSkew* since with amount of storage, the later one offers better performance.

5) Putting All Together - Realistic: Fig. 9 on the following page shows overall performance of all predictors in a realistic situation. We can see that *O-GEHL* is the best one among the four given that they all have 64KB of storage. The *TAGE* and *YAGS* predictors use tag to reduce aliasing, but that also uses precious storage resources. On the other hand *O-GEHL* and *2BcgSkew* do not use tag. *O-GEHL* is better than *TAGE* while *2BcgSkew* is better than *YAGS* in terms of MPKI.

6) Putting All Together - Ideal: Fig. 10 on the next page shows overall performance of all predictors. We can see that *O-GEHL* is still the best one among the four given that they all have 64KB of storage. The MPKI for *O-GEHL*

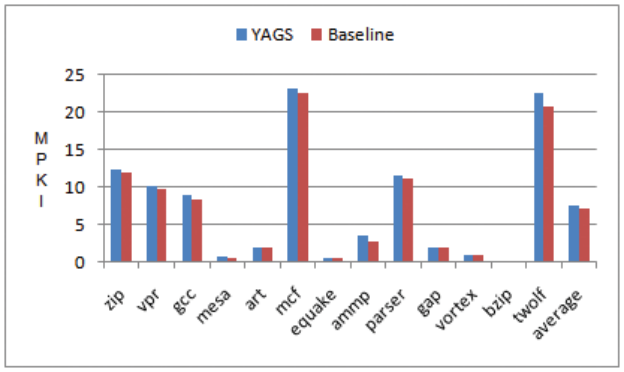

Fig. 8. YAGS Performance

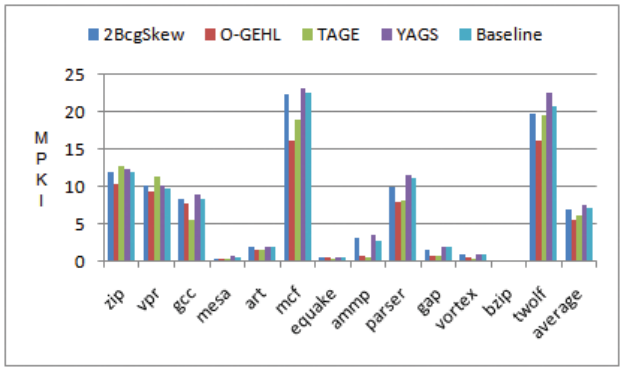

Fig. 9. Overall Performance Realistic

when running the trace to complete is 4.37. This number is also meaningful in the sense that it covers the whole execution of the benchmark and hides the learning time of the predictor. There are two test cases **mcf** and **twolf** which have higher MPKI compared with others. Simple statistic on the branches shows they have more irregular branches. More detailed analysis has to be performed to provide a solution to improve the performance.

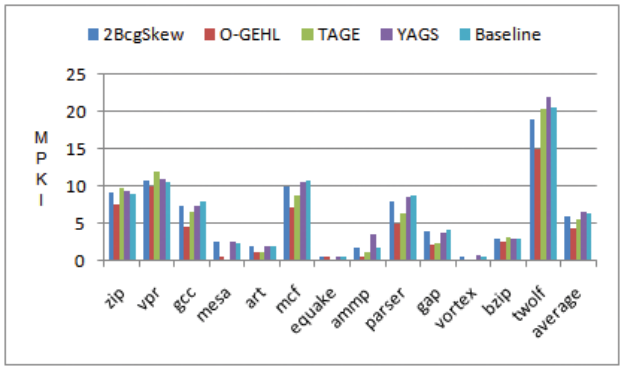

Fig. 10. Overall Performance Ideal

V. Conclusion and Future Work

Although it is expected that TAGE should be the best one according to previous work, the fact that it is also very hard to tune, especially the update policy. The best one in terms of average MPKI among the four predictors I obtained is O-GHEL. The evaluation of those predictor shows they have an average MPKI between 4 and 8 for the spec2k trace provided. Due to the short time frame of this project, there are still lots of improvement can be made. Future work including expanding TAGE predictor to L-TAGE and optimize it. Moreover, large predictor storage is conducive to the accuracy of the predictor. Thus, it is possible to virtualize it thus get better performance as in [8].

References

[1] A. Seznec, “A case for (partially) tagged geometric history length branch prediction,” Tech. Rep.
[2] A. Seznec and P. Michaud, “De-aliased hybrid branch predictors,” Tech. Rep.
[3] A. Seznec, “An optimized 2bcgskew branch predictor,” Tech. Rep.
[4] A. Seznec, P. Michaud, P. Michaud, R. Uhlig, and R. Uhlig, “The o-gehl branch predictor,” in *In Proceedings of the First Workshop Championship Branch Prediction in conjunction with MICRO-37*.
[5] A. Seznec, “Analysis of the o-geometric history length branch predictor,” in *ISCA ’05: Proceedings of the 32nd annual international symposium on Computer Architecture*, pp. 394–405.
[6] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge, “The bi-mode branch predictor,” in *MICRO 30: Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pp. 4–13.
[7] A. N. Eden and T. Mudge, “The yags branch prediction scheme,” in *In Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 69–77.
[8] I. Burcea, S. Somogyi, A. Moshovos, and B. Falsafi, “Predictor virtualization,” in *ASPLOS XIII: Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pp. 157–167.