Study on Loop Termination Buffer Potential Improvement

Bahar Ebrahimi
Departments of ECE
UWaterloo
Waterloo, Canada
b2ebrahi@uwaterloo.ca

Huadong Xing
Departments of ECE
UWaterloo
Waterloo, Canada
h24xing@uwaterloo.ca

***Abstract*—Speculating the termination of loops is one of the hard-to-predict scenarios in branch predictions. It is related to a large percentage of mis-predictions and causes performance degradation. In this report, we try to implement the dynamic level solution of one of the distinguished papers written in this topic [1] which uses an extension to primary BP for loop termination prediction. We implement the extension using simplescalar and aim to introduce some improvements to it. Moreover, a comprehensive literature review about the loop terminations is presented. Our simulation shows implementing the extension will increase accuracy of loop termination categories by 42% on average and up to 1.2% improvement on all branches.**

I. Introduction

By using deeper pipelines in newly designed architectures, the penalty of mis-prediction of a branch becomes larger and has a noticeable impact on performance. The speculated instructions in the wrong path should be flushed and instructions from the correct path will be started to be fetched and executed. This process results in consuming power on more instructions that were not supposed to be fetched and executed. Cause wasting the energy, other than degrading performance [2]. Predictions of a correct path for branches are important for both power and performance aspects. So, Branch predictors (BP) are used as one of the most important blocks to avoid stalls due to branch mis-prediction and improving performance [3].

Modern branch predictors have been able to achieve prediction accuracy of over 90% [1]. However, there are some special situations that are hard to predict with suggested methods and need to cope with extra hardware. These cases are responsible for most mis-predictions in branches. Researchers investigate these cases and mostly try to add side branch predictions to primary BPs. Side BPs are hardwares that specifically work well for some hard-to-predict scenarios and work as an extension to primary BPs.

Predicting the termination of loops, especially nested loops are one of the hard-to-predict cases that account for a large percentage of the branches that are predicted incorrectly and cause performance degradation. it is not possible to predict nested loop patterns easily. Specially nested loops with large number iteration cannot predicted by local or global BPs with bits smaller than iteration range [1].

In our project, we aim to implement a method proposed in a well known paper in this field [1]. it uses a Loop Termination Buffer (LTB) as an extension for loop cases to track termination patterns.

We are going to first implement the hardware solution proposed in the paper in a dynamic approach in simplescalar [4] and then make improvements upon it. Based on the original LTB, some of the features we are eager to investigate more are as follows. First, we like to explore the impact of different replacements on the prediction rate. Next, we look into the effect of buffer size on the accuracy of prediction.

The next sections in the report are as follows. First, in section II some information about implementation is presented, then we describe our methodology. in section IV the measurement will be illustrated. In section V a short literature review of papers working on loop termination is presented and finally, we provide a conclusion in section VI.

II. implementation

According to [1], the LTB buffer will work with the original Predictor when it encounters Loop Branch. At these times the PC Address is passed in both the LTB and Branch Predictor.

LTB contains a tag, several counters, and one confident bit. Where a tag is used to store the branch address, $SpecIter$ counter is used to store the loop count at the time of speculation, $non - SpecIter$ counter is used to store the real loop count after the branch is resolved and the $Trip$ counter is used to store the possible loop length. When the same loop length is found for second time, Confident Bit change to set.

In LTB, tag check to see whether the branch address is in the Buffer. When the address is found, the number of $SpecIter$ will be increased and compared with $Trip$. If they are equal and a confident bit is set, then output Loop Terminate, which overwrites the original predictor result. When a branch is resolved, the results are used to update the LTB. If a branch is never found, it will be added to the LTB. If the branch address already exists, depending on termination of loop $non - SpecIter$ will be update or $Trip$ resets. paper [1] has the process in more detail.

As the paper [1] suggested, Conditional Branch with a branch target address less than the current PC address will be considered as Loop. We understand that this might cause

some branches to be incorrectly classified as Loop Branch, but since we do not have a dedicated loop opcode, it is difficult to accurately identify Loop Branch. However, the main object is to find the number of branches that mis-predict with normal branch predictors but predict correctly with LTB. So, the accurate loop branch number had little influence on our conclusion.

The original LTB design did not specify the replacement algorithm to be used when the buffer is full, so we used the Round Robin (FIFO) algorithm as the baseline in the first step. To improve the implementation, we used LRU and Random in our code for the next step.

III. Methodology

In this project, the LTB algorithm was replicated in SimpleScaler [4] according to the paper [1] to verify the improved accuracy of branch prediction by LTB and the possible improvement of the experiment.

Simplescaler contains a build target called $sim - bpred$ and a build target called $sim - outorder$. $sim - bpred$ adds branch prediction to $sim - safe$, while $sim - outorder$ is an implementation of out-of-order execution. We implemented LTB in both programs, and consistent with our predictions, the two programs produced no difference in LTB statistics. Because LTB acts on the front end of the processor (Fetch, Issues), the stages that executed sequentially. And finally, we Update the LTB at the Commit stage, where the instruction has been reordered to maintain the von Neumann illusion.

several benchmarks from SPEC2006 Suit [5] like hmmer, mcf, gobmk alongside some other benchmarks like go, anagram and gcc was chosen for simulation in $sim - outorder$.

Simulation in cycle-accurate mode is slow. For modern suits with billions of instructions, like SEC2006, it can take months for simulating in this mode [6]. Several methods like fast-forward or using SimPoint tool used to make simulation faster by simulating only part of suit [6] [7]. In our simulation for hmmer and MCF fast-forwarding technique was used, as simulation of them was taking a long time. Paper [6] suggested that, fast-forwarding instructions around 1 billion and then started executing an interval instruction can be reasonable [6]. Some others suggest using 500 million for fast-forwarding [8]. Thus, for the hmmer benchmark, we used 1 billion for fast forward and then executed 500 Million instructions for two different input files. For MCF the benchmark was run two times with different fast forward values of 1 billion and 500 million, and both with 500 million instruction execution intervals.

Several stats printed, to investigate the effect of LTB and measure the outcome. The main recorded parameters came below.

- Numbers of loop branches,named as $ltb.loop\_branch$.
- Numbers of loop branches terminated, $ltb.loop\_term$.
- Numbers of loop branches terminated by normal branch predictor,named as $ltb.loop\_term\_hits\_bpred$.
- Numbers of loop branches terminated hit by LTB,named as $ltb.loop\_term\_hits\_ltb$.

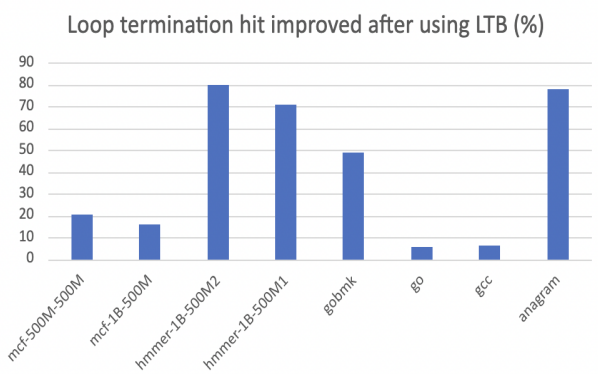


Fig. 1: Improvement of loop termination prediction after using LTB with size 8 in percentage(%).

- Loop Terminate Hit by LTB Only .Loop miss by normal branch predictor but hit by LTB. it is named as $ltb.loop\_term\_hits\_ltb\_only$.

.

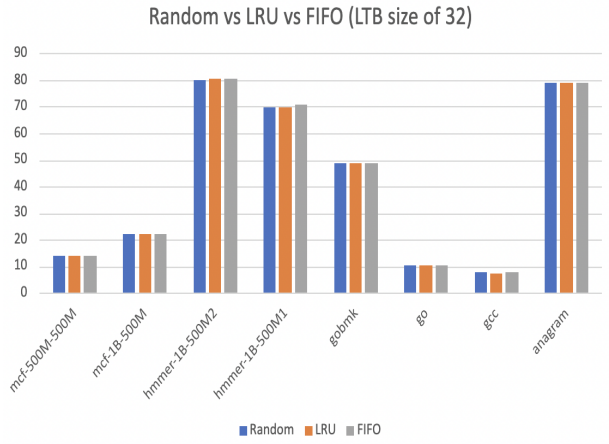
IV. Measurement

Benchmarks mentioned in previous section, ran for baseline code implements as paper [1]. The results compared with number of correct termination loops prediction with primary BPS and no LTB and presented in figure 1 . These numbers are useful to show whether the algorithm was successful to decrease mis-predictions in loop termination. Figure 1 show the improvements in prediction loop termination after using LTB with buffer size of 8 in the FIFO replacement algorithm. As we can see, using LTB can improve predicting loop termination up to 80% in benchmarks like hmmer from SPEC2006. The average improvement for all simulated benchmarks is 42% for loop termination category and overall prediction can improve up to 1.2%.

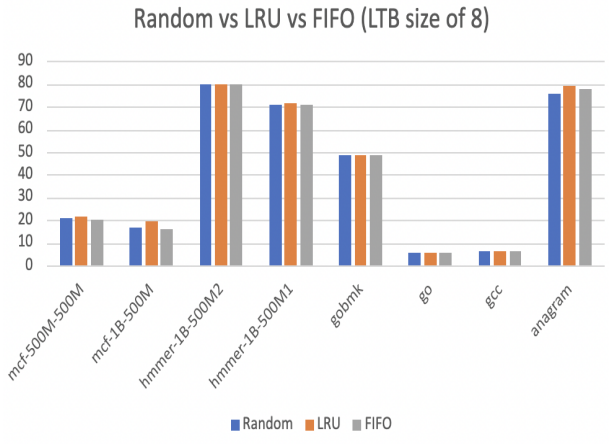
It can be seen that LTB is successful in detecting loop termination in comparison to primary BPs. Some benchmarks like gcc have smaller improvements rate number. We think one of the reasons, as we mentioned earlier, is in detecting loops correctly. It might detect some branches that are not loop as loops branch because their target address is backward. In these cases, branches are incorrectly categorized as loops while LTB cannot predicting it.

For the next step, in order to try to improve the prediction rate of the paper [1], we implemented several replacements algorithms such as LRU and RANDOM in extension to the existing FIFO replacement algorithm used in the previous step. Benchmarks simulated with all 3 algorithms. Figure 2 shows the improvement result of all 3 algorithms with different buffer sizes of 32 and 8.

It is observed from simulation results with buffer size 8, using LRU can decrease mis-speculation rate in small numbers. On average, using LRU as a replacement algorithm increases correct loop termination by 0.9% in buffer size of 8. However, with increasing buffer size, this number decreases. from figure 2 we can see that using LRU does not have



(a) With LTB Buffer size 32



(b) With LTB Buffer size 8

Fig. 2: Simulation result of different benchmarks for three different replacing algorithm

much impact. On the other hand, design of LRU in hardware is complex and has large overhead and power consumption. We think with the cost of designing and the fact that the improvement is small, using LRU for implementation does not worth it.

For the next step, we investigate the impact of different buffer sizes. different benchmarks simulated with buffer size varying from 8 to 256. figure 3 shows the improvement in predicting loop termination for different benchmarks with varying size for LTB buffer.

We found that the buffer size had no significant effect on the performance of LTB. Some benchmarks, such as GCC and hmmer-1B-500M2, even have diminishing returns. We suspect that the reason why the change is not significant, is that the with buffer size of 8 is still over kill for LTB. When programmers write a loop, they tend to pay attention to running it efficiently. They try to avoid too many nested loops. In general, three-layer nested loops are rare, so an LTB with a buffer size of 3 can handle most of the nested loops. Therefore,

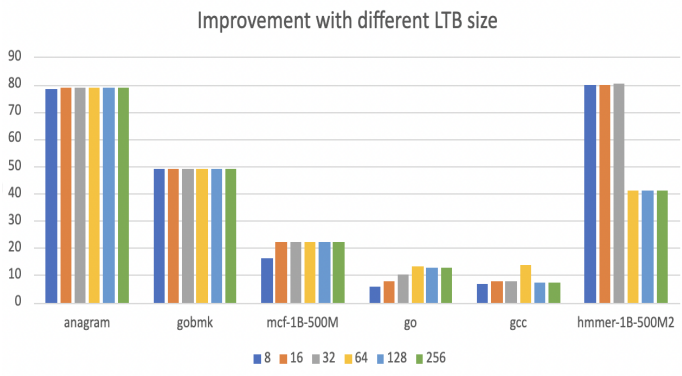


Fig. 3: Benchmarks with different LTB size

we assume that the buffer size of 8 is still too large for most cases. To verify this idea, we needed to test the conditions when buffer size of 1, 2, and 4. Unfortunately, we were not able to expand this experiment due to time constraints.

V. Related work

Predicting loop termination is an important topic. It can improve the performance and also the power consumption of the processor. Different research groups worked on the topic trying to improve the prediction from different aspects. Some of these papers will be reviewed in this section.

Sendag et al. [9] proposed a method that uses a complementary branch predictor (CBP) for the mis-predicted cases. A primary BP is used to generally predict the branches. CBP is used for scenarios with mis-prediction that are hard to predict for primary BP. CBP applies a branch mis-prediction predictor (BMP) that adopts a mis-prediction pattern related to an index. It is only dealing with indices that have mis-prediction and not all the branches. It keeps track of the distance between each mis-prediction. Distance number is then used to predict the next miss-prediction and consequently to ignore it. Since CBT is only employed in mis-predicted cases, it is not running at all times and does not consume much power consumption, Thus it is good for embedded device application. The method is good for early-loop exits and loops with a variant long number for iteration [10].

Other methods [11] suggest using a multi corrector for primary BP. Correctors, similar to what is mentioned in the previous paper, are used in parallel with primary BP for hard-to-predict cases where BP generates mis-predictions. They use confidence level to train the mis-prediction case and reverse the result in case of an incorrect decision made by primary BP. It uses 3 different correctors each of which is good for a specific kind of hard-to-predict branches. Correctors have their own confidence level and check their own branch patterns. Therefore, each one only checks their specific cases for mis-prediction. The interval corrector is designed to be used for predicted loop termination. It has an interval, confidence and counter value. The interval value is the trained value for prediction and confidence is used for showing the percentage of sureness. In each prediction based on the correctness of the

decision, the confidence level will be updated for that loop. Counter numbers have the number of correct predictions since the last mis-prediction. The other two correctors are bimodal and two-level correctors that are related to bimodal and two-level patterns, respectively.

Albericio et al. [12] also suggest side-predictors with the primary BP which is an ISL-TAGE predictor for finding loop termination in nesting loops. Like previous papers the primary branch is responsible for predicting the branch's outcome. In addition, the side-predictor is used for some specific scenarios that are hard to predict. Most BPs search in a linear continuous pattern to find the correlations in branches. However, in nesting loops the pattern is not linear most of the time and the branch output of an inner loop can correlate with the previous outer loop. So, these irregular patterns are hard to predict with primary normal BP and need large local history BPs. The paper suggests using a multi-dimensional matrix instead of a linear array can exploit those patterns like diagonal, by checking nesting loops history from different iterations. The paper uses a side predictor named wormhole with multi-dimensional histories. The slide-predictor will change the primary BP only if it is highly confident.

Hardware implementation of [12] is challenging and complex, so Seznec et al. [13], tried to use a multi-dimensional matrix concept for nesting loops with a simpler implementation. They use an Inner Most Loop Iteration counter (IMLI) that is the iteration number of the inside loop alongside the PC as an index to a table that will be used for the side predictor of the BP system. The tables exploit the correlations for multi dimensional patterns like [12]. The slide prediction can be used with any global primary BP like TAGE or perception BPs, a feature that is preferable in real hardware implementation [14]. While the previous paper uses local BP for slide prediction. Using this technique is more simple in implementation and has a smaller overhead.

[15] proposes a solution for branches with large loop iteration. As paper [1] suggested, local and global loops can not find loops that their iteration is larger than their bit range, so the paper uses a FAB predictor as a hybrid predictor along with other BPs. The FAB uses discrete frequency domain of branch outcome instead of time domain and results in saving longer iteration with less bit required. However, FAB does not work well with devotion from symmetric waves of branch outcome, half taken and half not-taken, since the frequency domain wave has many elements in different frequencies.

Some other studies find other solutions other than BPs [16] [17] [18]. Garcia et al. [16] try to solve the problem by adding a loop window in parallel to the reorder buffer ROB. When the processor reaches the loop instructions, the instructions that are related to the loop will be put in the loop window and will be sent for execution. The sequence of instructions in the loop window will be executed repetitively. The process does not need BP in these cases and does not use fetch, decode and reorder repeatedly, resulting in less power consumption. However, it cannot be used in nested loops or cases without one simple loop.

VI. Conclusion

As we observed from measurements, using LTB will increase the prediction rate. Loop terminations will be detected 40% more on average. With different replacement algorithm implementations that we add for improvements, we showed using LRU can increase prediction rate with with a small percent. However, implementation of LRU in hardware is hard and causes an increase in overhead and power consummation. Impact of using different LTB sizes were also investigated and effect on different benchmarks was presented. With having more time , these aspects could be checked with other more benchmarks and could have better insight.

The other feature that can be investigated for future work is to use LTB only at high ratio mis-prediction cases in loop termination. In the current implementation suggested by paper [1], all loops were updated in LTB parallel with primary BPs. LTB is updating even in loops that primary BPs are successful in prediction. it results in more power consumption. we can use a confidence level for loop termination and use it to decide when LTB is needed. Whenever there is a high rate mis-prediction for loop termination, LTB can interfere and for cases where primary BPs are predicted correctly, we don't need to waste power on LTB. Using this method can cause less power consumption, however, the overhead for designing a confident level should be considered.

References

[1] Sherwood, Timothy, and Brad Calder. "Loop termination prediction." In International Symposium on High Performance Computing, pp. 73-87. Springer, Berlin, Heidelberg, 2000.

[2] Parikh, Dharmesh, Kevin Skadron, Yan Zhang, Marco Barcella, and Mircea R. Stan. "Power issues related to branch prediction." In Proceedings Eighth International Symposium on High Performance Computer Architecture, pp. 233-244. IEEE, 2002.

[3] Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.

[4] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," ACM SIGARCH Computer Architecture News, June 1997

[5] Standard Performance Evaluation Corporation. SPEC CPU benchmark suite (2006), http://www.specbench.org/osg/cpu2006/

[6] Weaver, Vincent M., and Sally A. McKee. "Using dynamic binary instrumentation to generate multi-platform simpoints: Methodology and accuracy." In International Conference on High-Performance Embedded Architectures and Compilers, pp. 305-319. Springer, Berlin, Heidelberg, 2008.

[7] Yi, J., Kodakara, S., Sendag, R., Lilja, D., Hawkins, D.: Characterizing and comparing prevailing simulation techniques. In: Proc. 11th IEEE Symposium on on High Performance Computer Architecture, pp. 266–277 (February 2005)

[8] SPEC CPU2006 Compilation for SimpleScalar/Alpha-OSF http://www2.ece.rochester.edu/ parihar/spec2k6.html

[9] Sendag, Resit, J. Yi Joshua, Peng-fei Chuang, and David J. Lilja. "Low power/area branch prediction using complementary branch predictors." In 2008 IEEE International Symposium on Parallel and Distributed Processing, pp. 1-12. IEEE, 2008.

[10] Mittal, Sparsh. "A survey of techniques for dynamic branch prediction." Concurrency and Computation: Practice and Experience 31, no. 1 (2019): e4666.

[11] Lai, Chunrong, Shih-Lien Lu, Yurong Chen, and Trista Chen. "Improving branch prediction accuracy with parallel conservative correctors." In Proceedings of the 2nd conference on Computing frontiers, pp. 334-341. 2005.

[12] Albericio, Jorge, Joshua San Miguel, Natalie Enright Jerger, and Andreas Moshovos. "Wormhole: Wisely predicting multidimensional branches." In 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 509-520. IEEE, 2014.

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

[14] Seznec, André, Stephen Felix, Venkata Krishnan, and Yiannakis Sazeides. "Design tradeoffs for the Alpha EV8 conditional branch predictor." ACM SIGARCH Computer Architecture News 30, no. 2 (2002): 295-306.

[15] Kampe, Martin, Per Stenstrom, and Michel Dubois. "The FAB predictor: Using fourier analysis to predict the outcome of conditional branches." In Proceedings Eighth International Symposium on High Performance Computer Architecture, pp. 223-232. IEEE, 2002.

[16] Garcia, Alejandro, Oliverio J. Santana, Enrique Fernández, Pedro Medina, and Mateo Valero. "LPA: A first approach to the loop processor architecture." In International Conference on High-Performance Embedded Architectures and Compilers, pp. 273-287. Springer, Berlin, Heidelberg, 2008.

[17] Thornton, James E. "Parallel operation in the control data 6600." In Proceedings of the October 27-29, 1964, fall joint computer conference, part II: very high speed computer systems, pp. 33-40. 1964.

[18] Vajapeyam, Sriram, and Tulika Mitra. "Improving superscalar instruction dispatch and issue by exploiting dynamic code sequences." ACM SIGARCH Computer Architecture News 25, no. 2 (1997): 1-12.