VIP Branch Predictor: Improving Branch Prediction Accuracy with Private Predictors and Miss Cache

Zerui An
Electrical and Computer Engineering Department
University of Illinois at Urbana-Champaign
Urbana, USA
zeruian2@illinois.edu

Yichi Zhang
Electrical and Computer Engineering Department
University of Illinois at Urbana-Champaign
Urbana, USA
yichi2@illinois.edu

***Abstract*—Modern branch predictors such as L-TAGE [1] and perceptron [2] can achieve high prediction accuracy in most workloads. However, we have noticed that these predictors usually struggle with a few hard-to-predict branches in each workload, which contribute to the majority of mispredictions. In this paper we propose a branch predictor design that can identify these hard-to-predict branches using a miss cache and increase their accuracy using private predictors[1].**

***Index Terms*—branch prediction, L-TAGE predictors, perceptron predictors**

I. Introduction

In order to better understand the accuracy bottleneck of modern branch predictors, we profiled L-TAGE [1] and perceptron [2] predictors and find out that they are struggling with just a few hard-to-predict branches. To solve this, we came up with a cache-based design to identify these branches and assign each one of them a private branch predictor to remove any possible interference from other branches.

In this paper, we will discuss two of the state-of-the-art branch predictors, L-TAGE and perceptron in section II. In section III, we will discuss some related work on solving branch prediction problem. In section IV, we will explain our motivation with data. In section V, we will explain our design in detail. In section VI, we will explain our methodology. In section VII, we will evaluated our model using DPC-3 [3] traces and present our results. Finally, we will conclude this paper in section VIII.

II. Background

A. Perceptron Branch Predictor

The perceptron predictor [2] was proposed in 2001 and has been used widely in processors since then. To make a prediction, a vector of weights is first retrieved from the perceptron table using the branch PC as index. Then, a vector multiplication is computed using the weight vector and the branch history (a "taken" branch is considered a 1 and a "not-taken" branch is considered a -1). The final prediction is made based on the sign of the vector multiplication result. Figure 1 shows a high-level block diagram of a perceptron predictor.

[1]https://github.com/JerryAZR/SandboxPredictor

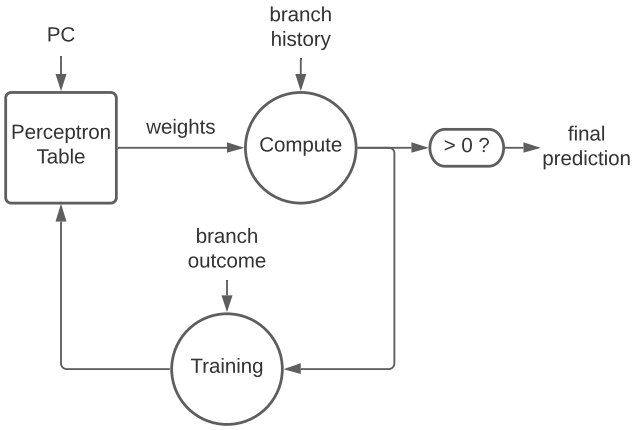

Fig. 1. A high-level block diagram of a perceptron predictor

B. L-TAGE Branch Predictor

The L-TAGE branch predictor [1] combines a TAgged GEometric history length (TAGE) predictor [4] and a loop predictor. By default, predictions are provided by the TAGE predictor. However, when a regular loop with constant number of iterations is identified, the prediction made by the loop predictor would be used instead. Figure 2 illustrates an L-TAGE predictor combining a two-component TAGE predictor and a loop predictor.

III. Related Work

Similar to our findings, the observation made by Chit-Kwan Lin and Stephen J. Tarsa [5] also shows that the bottleneck of branch prediction lies in a few hard-to-predict branches. Instead of improving the current dynamic branch predictors that are trained "online" (i.e. during run-time) with branch history, they proposed training the predictors "offline" and making use of more sophisticated machine learning models and information other than branch history. While their proposed solution may be able to achieve near-perfect or even perfect branch prediction accuracy, it would require a significant amount of training data and possibly some major changes to how branch predictors interact with the rest of the components. In contrast, the design we propose in this paper

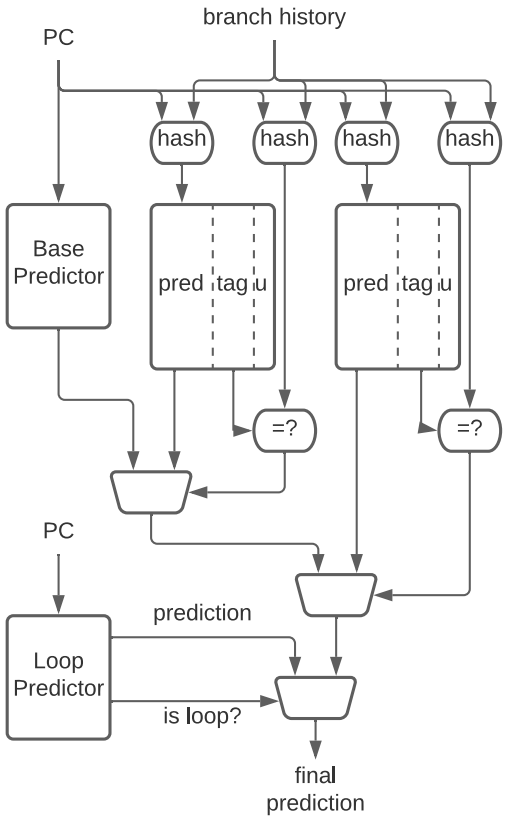

Fig. 2. An L-TAGE predictor using a two-component TAGE predictor and a loop predictor (update logic not included). The final prediction is selected between the output of the TAGE predictor and the loop predictor based on the confidence of the loop predictor

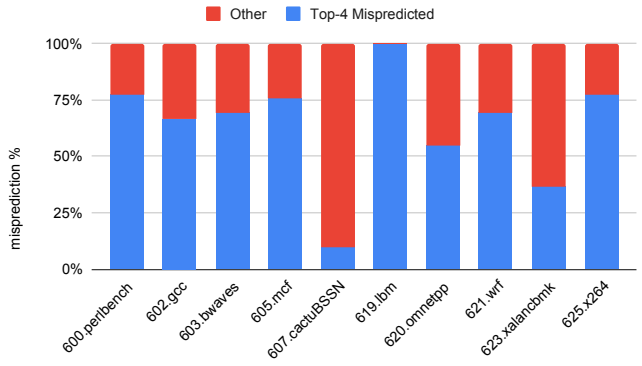

Fig. 3. Percentage of mispredictions contributed by the top four mispredicted branches

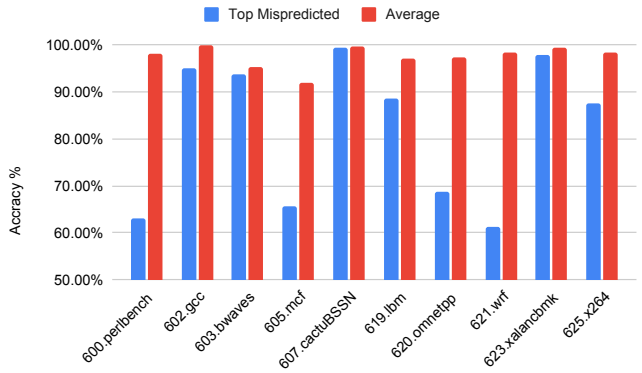

Fig. 4. A comparison between the average branch prediction accuracy and the accuracy of the top-mispredicted branch

shares the same interface and assumptions with existing branch predictors. Therefore, we believe that our proposed design can be integrated into modern processor designs with minimal modification.

IV. Motivation

By analyzing the branch misprediction patterns, we have observed that in most of our benchmark programs, there exists a few hard-to-predict branches that contribute to the majority of the mispredictions. Figure 3 shows the contribution of the top four mispredicted branches to the total misprediction count. We could see from the figure that in most of the benchmarks over 50% of the mispredictions are contributed by only four branches. The data was collected using a 4 kB L-TAGE predictor. While the exact numbers in other predictors may not be the same, the overall distribution is similar. These branches also have a lower prediction accuracy than the average. From Figure 4 we can see that the difference is usually above 5% and sometimes even up to 30%.

Based on the results we have collected, we conclude that if one wants to improve the overall accuracy of branch prediction, then a method to accurately predict hard-to-predict branches must be found. As suggested in [5], simply increasing the size of existing branch predictors is not very efficient. Therefore, we plan to separate these branches from regular ones and use dedicated resource to improve their prediction accuracy.

V. Design

Our plan to increase branch prediction accuracy can be summarized into two goals:

1) Separate hard-to-predict branches from regular branches
2) Increase the accuracy of these hard-to-predict branches

We use misprediction frequency rather than prediction accuracy to identify hard-to-predict branches. Because we believe that rather than focusing on low-frequency branches that do not cause many mispredictions, it is more valuable to work on improving the prediction accuracy of frequently mispredicted branches, even if they have higher base accuracy.

A cache identifies frequently accessed memory addresses that are likely to be accessed in the future. This property is very similar to what we want to achieve in our first goal, so we built a cache-like structure to identify frequently mispredicted branches. To achieve our second goal, we assign a private branch predictor to each identified hard-to-predict branch. Unlike the base predictor built for all regular branches that

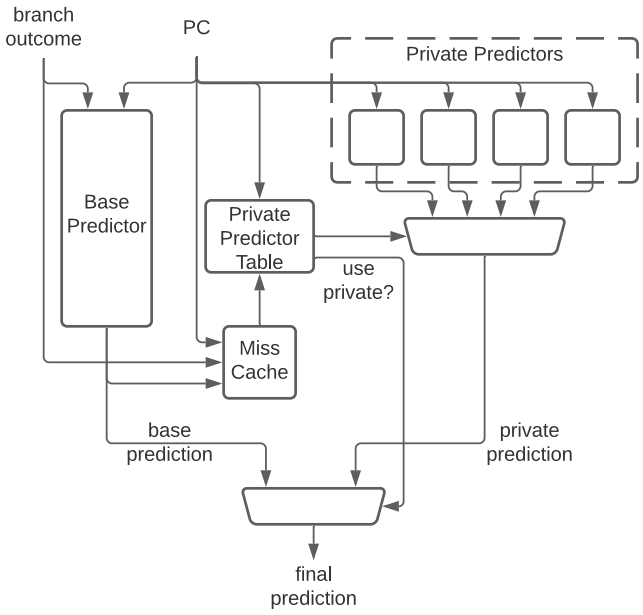

Fig. 5. Our proposed branch predictor with a base predictor and four private predictors

only uses global history, the private predictors also use local history when making predictions and training.

Figure 5 shows a block diagram of our proposed predictor design. By default, predictions are made by the base predictor. Whenever the base predictor mispredictions, the miss cache is accessed using the branch PC. The miss cache generates a private predictor table which maps each frequently mispredicted branch to one of the available private predictors. If a branch hits in the table, the prediction from the corresponding private predictor will override the default prediction. We name this design the "VIP predictor" because there are a few branches in every program (the "VIPs") that are assigned a private predictor each, while all other regular branches have to share the base predictor.

A. Private Predictor

Because there are multiple private predictors in our proposed design, the size of each predictor must not be too large. A history-indexed branch predictor (e.g. gshare) cannot be used here because each additional history bit would double the size of the predictor. On the other hand, a perceptron is a natural fit for a private predictor because only one set of weights is needed for a given branch. The size of a perceptron scales linearly with the history length, so it is easy to add history bits if necessary.

As stated in [2], a single perceptron cannot predict "non-linear" branches accurately (e.g., some branch that is "taken" if and only if the last two branches have the same outcome). To minimize the impact of these branches, we choose an L-TAGE predictor as our base predictor. We expect the L-TAGE base predictor would be able to accurately predict most of the branches that the private perceptron predictors do not excel at.

B. Miss Cache

The miss cache is a "tag-only" cache that is accessed when the base predictor mispredictions. Ideally, after a program runs for a while, only the frequently mispredicted branches remain in the miss cache. These branches are then mapped to available private predictors. In reality, the cache content may change frequently as the programs runs, especially if the replacement policy is not very good. This frequent change in cache content could lead to the branch-to-private-predictor mapping being altered frequently, which in turn could reduce the accuracy of private predictions. To avoid this situation, we propose two solutions.

First, we could take snapshots of the cache content on fixed intervals and use them to generate the mapping. This way, the mapping is not affected even if the cache content changes. This method does not rely much on the quality of the replacement policy and can guarantee the stability of the mapping within a period. The downside is that any changes in the program behavior are not seen by the private predictors until the next period, so the private predictors may not be able to adapt to the new pattern in a timely manner.

The second solution we proposed is to avoid using replacement policies like least-recently-used (LRU) or not-most-recently-used (NRU) because these policies always creates a new entry (and possibly evicts an old one) when a new access is seen. To make things worse, the inserted entry is always marked as the most-recently-used until the next access, making it nearly impossible to tell if an entry is frequently accessed. Instead, we would like to use replacement policies that are able to capture how frequently an entry is accessed, because they would generally lead to a "stable" cache content (i.e. at least part of the content doesn't change frequently).

These two solutions can be combined. In our design, we use a segment LRU cache as our miss cache and only the protected segment is used for snapshots and creating the private predictor table.

C. Private Predictor Table

The private predictor table is created by the miss cache. It maps frequently mispredicted branches to available private predictors. However, because "frequently mispredicted branches" do not always have low prediction accuracy, it is possible for the private predictors to perform worse than the base predictor, especially if the private predictors are perceptron-based and the branch is "non-linear". To avoid performance loss in such situations, we use a competition-based strategy to choose between the base predictor and the private one (if available). When predicting the direction of a branch in the private predictor table, the table first checks if the prediction history favors the private predictor. Only if it does, the table would select the private predictor as the prediction provider.

VI. Methodology

We have implemented our predictor design in ChampSim [6], a trace-based architectural simulator that models an out-

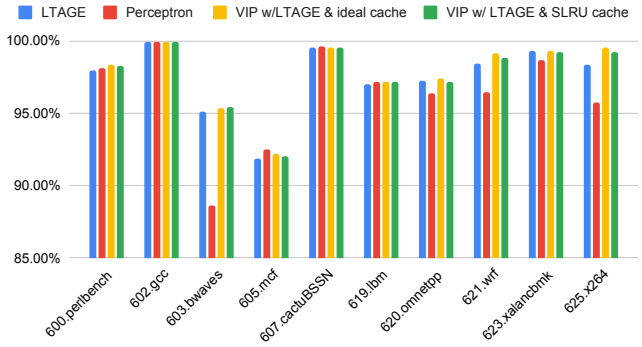

Fig. 6. Overall accuracy of the perceptron, L-TAGE, and VIP predictors

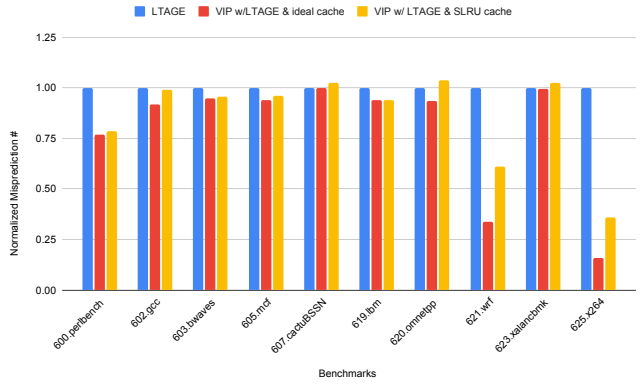

Fig. 7. Hard-to-predict branch accuracy of the perceptron, L-TAGE, and VIP predictors

of-order core. To perform roof-line analysis on the miss cache, we have also built an "ideal cache" that contains hard-coded addresses of the frequently mispredicted branches in each program. We then compare our design to an L-TAGE predictor and a perceptron predictor and present the results in the next section. In order to compare all predictors on a level ground, all predictors used for evaluation are sized to have 4 kB storage.

VII. Evaluation

For our evaluation, we simulated a single out-of-order superscalar core with L1, L2 and LLC caches. We used LRU as our LLC replacement policy and no prefetching for our caches. We used DPC-3 [3] traces as our benchmarks.

As shown in Figure 6, we can see some improvement in overall branch prediction accuracy in most benchmarks when compared to L-TAGE. In most cases, VIP predictors with SLRU caches achieved similar accuracy to VIP predictors with ideal cache. We can see that our predictor performs exceptionally well on wrf, x264 and perlbench benchmarks. This is associated with the fact that these benchmarks has just a few hard-to-predict branches that contributes to the most of the total mispredictions as shown in Figure 3. This allows our miss cache to correctly and consistently identify

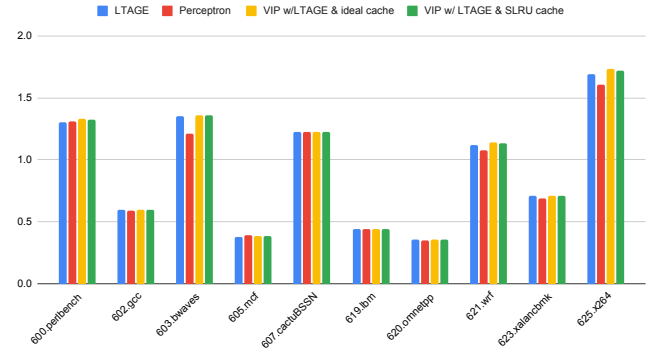

Fig. 8. IPC of the system using the perceptron, L-TAGE, and VIP predictors

these branches as hard-to-predict and keep them inside their own private predictors most of the time, avoiding most of the interference from other branches.

Regarding hard-to-predict branches, our predictor reduces the number of mispredictions by a considerable amount compared to L-TAGE. Unsurprisingly, predictors with ideal cache have better performance overall, which suggest a potential improvement in our miss cache. This is due to the fact that our miss cache is not perfect and it will occasionally put non-hard-to-predict branches into private predictors and lower its accuracy due to interference. Consistent with the last paragraph, our predictor has very high improvement in perlbench, wrf and x264 for the same reason.

To access the performance impact our branch predictor will bring to the whole architecture, we measured the IPC of our system in Figure 8. As we can see, in most cases, our branch predictor has similar IPC to L-TAGE and perceptron predictors. This is not ideal as we would like to see some improvement on IPC due to less pipeline flushes. This is because our improvement in accuracy is simply not significant enough to cause a considerable impact on the IPC.

One exception benchmark here is the Ibm benchmark where there is only one hard-to-predict branch that takes up all the mispredictions in the trace. One would expect our predictor to perform well on this benchmark, but Figure 6 shows minor improvement in overall accuracy and Figure 7 shows this hard-to-predict branch received only moderate improvement. This is probably because this branch is input data dependent as all predictors performs almost as bad and having its own private predictor at all time does not improve its accuracy too much.

Another interesting fact we noticed here is that some benchmarks showed patterns where perceptron predictor performs much worse than other predictors. This is possibly due to non-linear branch histories which makes it hard for perceptrons to model. Since we are using perceptrons as our private branch predictors, we implemented the competition-based decision scheme in the private predictor table to prevent these branches from negatively affecting the overall accuracy.

VIII. Future Work

As we have shown in the previous section, there are many possible improvement to our predictor. We could further improve our miss cache by using RRIP [7] replacement policy to prevent non-hard-to-predict branches from entering private predictors. We could also improve our private branch predictors by using more sophisticated design such as multi-level neural networks mentioned in [8] and [9].

IX. Conclusion

In this paper, we presented our VIP branch predictors which uses a miss cache to identify hard-to-predict branches and uses private branch predictors to minimize interference. In section VII, we showed that our design provide some improvement to prediction accuracy using DPC-3 [3] traces without using more storage. We also showed that our predictor successfully reduce the amount of hard-to-predict branch mispredictions in most cases. Our VIP branch predictor with SLRU caches also archived similar performance to a VIP branch predictor with ideal cache. Although the improvements are minor in many of our benchmarks, we still believe that we made a step in the right direction by specifically targeting the hard-to-predict branches.

Acknowledgment

We thank all staff and faculty members of ECE 511, Dongkai Wang, Nathaniel Bleier and professor Rakesh Kumar for all the assistance throughout the fall 2021 semester.

References

[1] A. Seznec, "The l-tage branch predictor," *J. Instr. Level Parallelism*, vol. 9, 2007.

[2] D. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pp. 197–206, 2001.

[3] "Dpc-3." DPC-3. https://dpc3.compas.cs.stonybrook.edu/.

[4] A. Seznec and P. Michaud, "A case for (partially) tagged geometric history length branch prediction," *Journal of Instruction-level Parallelism - JILP*, vol. 8, 02 2006.

[5] C.-K. Lin and S. J. Tarsa, "Branch prediction is not a solved problem: Measurements, opportunities, and future directions," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pp. 228–238, 2019.

[6] "Champsim." GitHub. https://github.com/ChampSim/ChampSim.

[7] A. Jaleel, K. B. Theobald, S. C. Steely, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *SIGARCH Comput. Archit. News*, vol. 38, p. 60–71, jun 2010.

[8] S. Zangeneh, S. Pruett, S. Lym, and Y. N. Patt, "Branchnet: A convolutional neural network to predict hard-to-predict branches," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 118–130, 2020.

[9] Y. Mao, H. Zhou, X. Gui, and J. Shen, "Exploring convolution neural network for branch prediction," *IEEE Access*, vol. 8, pp. 152008–152016, 08 2020.