Ensemble Learning in Branch Prediction

Umur Darbaz (darbaz2), Vikram Sharma Mailthody (vsm2), Andrew Pensinger (andrewp5)

ABSTRACT

Branch predictors are responsible for learning the execution trace of a program. This allows instructions to be speculatively fetched. Higher accuracy branch prediction schemes result in faster fetching. A limitation of traditional prediction schemes is the limited learning capabilities. Recent research in machine learning has shown that an ensemble of learners can outperform any single learner. This is dependent on the correlation levels of learners, in addition to selection and learning policies. The Tournament branch predictor is a simple ensemble which follows a set policy to select between and update them. We observed the possible weaknesses of the discrete-selection approach used in Tournament, where there is a learner with correct prediction up to 73 percent of the time in some benchmarks. Our approach to ensemble learning aims to utilize the correct information which discrete selection dismisses. Our results emphasized the importance of three factors in ensemble learning and branch prediction. These are; (1) learning/unlearning rate, (2) selection schemes and (3) correlation levels between learners. The ideal ensemble should be able to adapt rapidly, prevent noise in selection, and prevent uniform learning across the selected experts. We highlight the strengths and drawbacks of the discrete selection scheme, and investigate two potential implementations of an ensemble.

1. INTRODUCTION

Accurate branch predictions is essential to the performance of modern processors. A branch misprediction can cause a pipeline flush, which leads to stalls and higher energy consumption. For decades, researchers have proposed various techniques like Tournament, YAGS, L-TAGE Branch Predictor to improve the prediction accuracy [1][2][3]. One of the most effective branch prediction methods is to combine multiple branch predictors into a hybrid predictor or tournament branch predictor using discrete selection scheme. Here, the discrete selection is based on a master algorithm whose sole responsibility is trace past performance of the experts and select the expert which performs the best over a period of time. One of the problems with discrete selection is inductive predilection caused by over-fitting of the master algorithm.

On a closer observation, the selection scheme is essentially a learning problem. For each trail of selection, the master algorithm does a prediction and at the end of the trail, the actual outcome is used to update the master algorithm for future predictions. All predictions and outcomes are binary stream of information and the master algorithm learns over a period of time who needs to be selected. Recent research in machine learning has shown that an ensemble of learning algorithms can yield better results than a single learning algorithm or selection based scheme [4]. The tournament branch predictor uses a simple form of ensemble learning by selecting between two branch predictors. In this report, we investigate the possible weaknesses of the selection based approach and evaluate weighted accumulation selection scheme. We report our analysis based on Tournament Branch predictor.

In the next section, we discuss prior work. Section 3 provides detailed evaluation of selection based tournament branch predictor. Followed up discussion of design methodology and results for ensemble selection scheme.

2. RELATED WORK

Accurate Branch Prediction research has been around for decades. The most relevant work to this paper falls into two categories. First, past research based on combing multiple experts in discrete selection scheme. Here, a master algorithm, who is trained solely based on past performance, is used to perform the discrete selection among multiple experts. The current predictions by the experts does not have influence in the final decision process. One example of this strategy is Tournament Branch Predictor [1]. This predictor selects one of the two experts using discrete selection.

The high level view of two expert tournament selection based logic is shown in the figure 1. The local and global experts each make their own prediction, then the choice predictor (or the master algorithm) uses expert selection technique, i.e final prediction is formed by choosing one of the two experts solely on the past performance of the experts. To measure the performance of the experts, tournament branch predictor uses 2-bit counters. The implementation is similar to 2-bit local counter based branch prediction. However, in tournament, it uses outcome of branch prediction instead of instructions.

Second category fall under a more recent research area machine learning and specifically work on weighted majority based selection scheme. Perceptron branch predictor used weighted accumulation scheme of machine learning where final outcome is based on weighted majority vote of past inputs. However, first known work on weighted majority algorithm was first by Vovk in [5]. However, detailed analysis of Weighted Majority algorithm was implemented and named by Littlestone and Warmuth in 1994. In [6], Littlestone and Warmuth discussed used Weighted Majority algorithm for

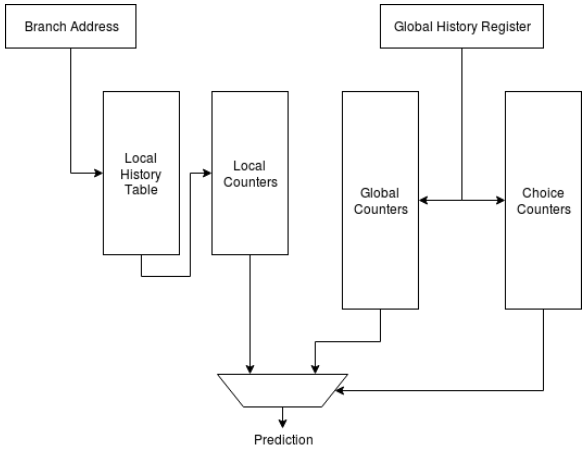

Figure 1: Tournament branch predictor

binary prediction of sequences and prove that the weighted majority algorithm is capable of predicting which experts perform well without the prior knowledge. The algorithm is widely used in language processing, gene predictions, etc. [4][7]

3. MOTIVATION

The initial motivation for the project was from [4]. Here, branch predictor used a weighted accumulation of 3 to 6 experts, depending on the size budget. The predictor uses a different weight update scheme from the one discussed in this report.

Another motivation for the ensemble branch predictor is the percentage of "correctable" mispredictions in our simulations of the tournament branch predictor. A "correctable" misprediction is a misprediction where one of the sub-predictors had the correct prediction, but was not selected. We believe that there may be a superior selection mechanism that can recover some of these "correctable" mispredictions. Figure 2 shows the percentage of "correctable" mispredictions from the tournament branch predictor simulations. These percentages are an upper bound on the number of mispredictions that an alternative selection scheme can recover.

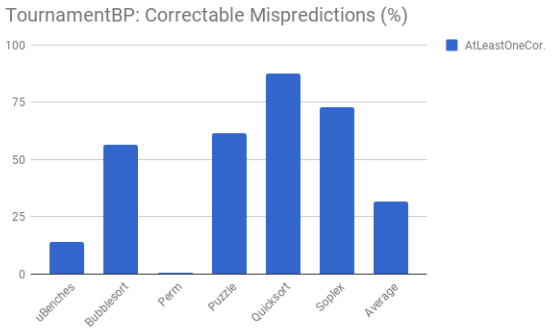

Figure 2: Correctable branches in tournament branch predictor

This work is primarily motivated by the problem that we anticipated to see in discrete selection scheme with multiple experts. It is not our primary goal here to build a state of the art branch predictor, but rather evaluate alternate avenue apart from discrete selection scheme.

4. PROJECT DESCRIPTION

Our project evaluates whether an alternative selection mechanism can outperform the discrete selection of the experts in the tournament branch predictor. Our alternative selection method is based on the weighted majority algorithm (WMA) [6]. This approach assigns a weight to each expert in the learning algorithm. The decisions of the experts are multiplied by their respective weights and accumulated. The direction of predictions determines the signs of inputs to the accumulated verdict. The accumulation is used to determine the final decision. For binary predictions, the accumulation is compared with a threshold to determine the final prediction value. Equation 1 shows WMA for an n-expert binary prediction, where p_i and w_i are the prediction and weight for expert i, respectively. We evaluate the viability of an alternative selection scheme compared to the discrete selection used in Tournament Branch Prediction. Our scheme is based on weighted accumulation [6]. This approach assigns and dynamically adjusts weights to each expert in the ensemble. Decisions made by experts are multiplied by their predictions and accumulated. The direction of the prediction determines the sign of the inputs to accumulation. The accumulation and a static threshold are used to determine the final verdict, the prediction.

$$\text{prediction} = \text{threshold} < \sum_{i=1}^{n} p_i * w_i \qquad (1)$$

Our implementation of the weighted majority algorithm uses n-bit saturating counters for the weights. The weight counters are updated after the actual outcome of the branch is determined. The sum of the weights are fixed to a certain value in the ensemble, in order to maintain an acceptable adaptation rate. The adaptation rate is bound by two variables; the learning rate and unlearning rate. On a high level, this is the rate at which each individual expert learn a new branching pattern. At the implementation side, this is made up of two components. The first is the weight increment/decrement rate, and the second is the predictor counter increment/decrement rate along with their respective depths.

An expert's weight is updated under two circumstances. First, if the expert voted in the same direction as the final prediction. Second, if the sum of weights has exceeded the aforementioned fixed value and the expert voted in the opposite direction as the final prediction. Under the first condition, an expert will see a weight increase if the final prediction was correct, and it will see a weight decrease if the final prediction was incorrect. For the second condition, an expert will see an adjustment to their weights in the following way: if the final prediction was correct, the expert's weight will be decremented. If the final prediction was incorrect, the expert's weight will be incremented. The weight adjustment rate is as follows: if one expert's weight is updated under the first rule, it will have an update rate of 2 and the adjustment rate will be 1 for the other experts in the ensemble. If two

experts had their weights updated under the first rule, they will have an update rate of 1 and the adjustment rate will be 2 for the other experts in the ensemble. The weights are not updated if all experts voted the same way.

The prediction counter updating is straightforward. If the expert's prediction is correct, their predictor counter is incremented. Else, it is decremented. The counter is updated regardless of the ensemble verdict.

These policies are shown in the pseudo-code below.

```
for predictor in ensemble begin
    # Update prediction counters
    if verdict == taken begin
        predictor[idx].counter++
    end
    else begin
        predictor[idx].counter--
    end

    # Determine learning and adjustment
    rates
    if contributorCount == 1 begin
        lrnRate = 2
        adjRate = 1
    end
    else if contributorCount == 2 begin
        lrnRate = 1
        adjRate = 2
    end
    else begin
        lrnRate = 0
        adjRate = 0
    end

    # Update and adjust weights
    if predictor.vote == verdict begin
        if verdict == outcome begin
            predictor.weight += lrnRate
        end
        else begin
            predictor.weight -= lrnRate
        end
    end
    else if weightSum > MAX_WEIGHT_SUM begin
        if predictor.vote != verdict begin
            if verdict == outcome begin
                predictor.weight -= adjRate
            end
            else begin
                predictor.weight += adjRate
            end
        end
    end
end
```

Algorithm 1: Update policies

For our baseline, we built and validated a three-way discrete selection predictor (figure 3). GShare, Local and Global predictors were combined in the same design as Tournament Branch Prediction, with one-third range for each predictor. This was achieved by using two choice counters and two thresholds.

The first ensemble design (figure 4) extended the three-way discrete selection by replacing the two choice counters with a weight table per expert and an accumulation scheme with a threshold to determine the final prediction. The weight tables were set to the same size as the predictor counter tables. The indexing scheme is identical to the predictor indexing scheme for each expert. This provided each predictor counter with an individual weight.

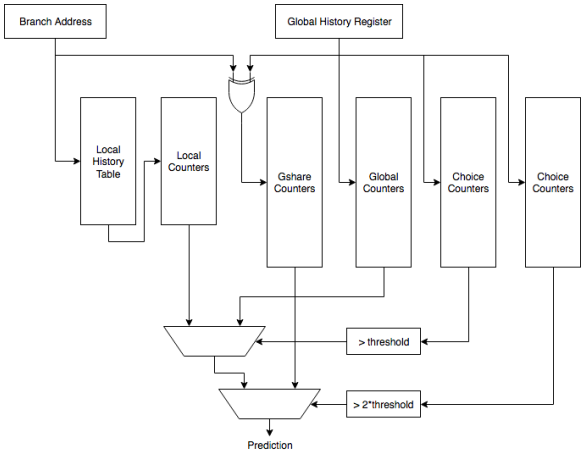

Figure 3: Baseline 3-way tournament predictor

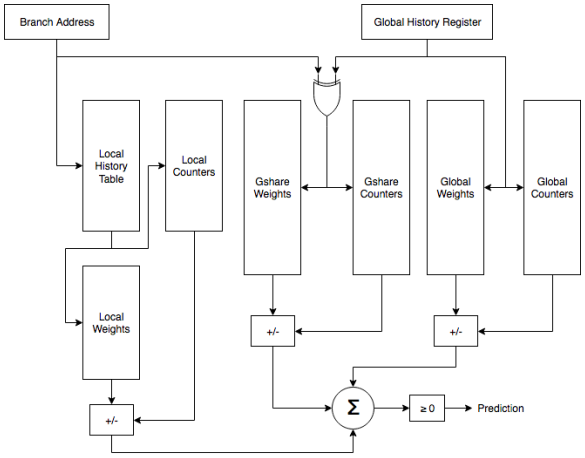

Figure 4: Ensemble with weight tables

The second ensemble design (figure 5) reduced hardware overhead by using a single weight per expert. The sum of the weight registers were fixed to a predetermined constant in this design. As previously described, this is to integrate the baseline design's adaptation speed while preserving the ensemble accumulation. This design is the final step in our ensemble design iteration. In the next section, we explain our simulation and analysis methodology.

5. METHODOLOGY

The branch predictor designs were simulated using the Gem5 simulator [8]. The designs built upon the tournament branch predictor is included with Gem5, which models the Alpha 21264 tournament branch predictor.

The branch predictors were simulated using four sets of benchmarks. The first set of benchmarks are microbenchmarks. These small programs test branch predictor performance for particular branching patterns. The next set of benchmarks were the "kill" benchmarks, which attempt to expose weaknesses in the tournament branch predictor. The third set of benchmarks were from LLVM [9], namely bubble-

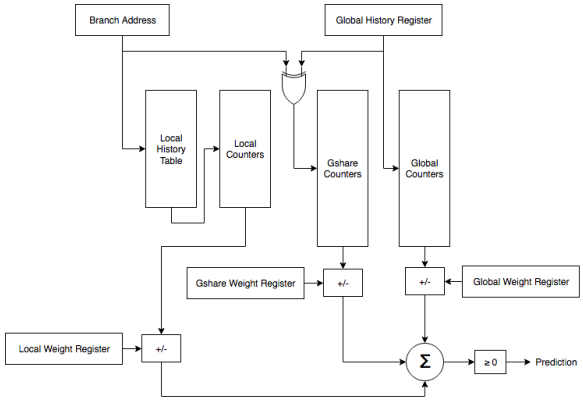

Figure 5: Ensemble with weight registers

sort, quicksort, perm, and puzzle. The final set of benchmarks were soplex and blackscholes benchmarks from SPEC.

The microbenchmarks and "kill" benchmarks make use of `libm5`. The `m5_reset_stats` function was used before the critical section of the benchmark. This function resets the values for all of Gem5's statistics. This allows for easier analysis of the predictor's performance for the benchmark's branching pattern, since the statistics collected while loading and initializing the benchmark program are not included. The reset command allowed for the reduction of Gem5's overhead. Gem5 does 15-17 look-ups per if conditional, which is a significant overhead for microbenchmarks.

We added new statistics to Gem5 to track the performance of the branch predictor designs. These statistics—in addition to accuracy—align with our defined taxonomy. Namely, we tracked:

1. The number of times there was at least one correct expert on a misprediction.
2. When all experts voted incorrectly.
3. When all experts voted correctly.
4. Number of times when there were one or two correct experts on mispredictions.
5. The number of times two lower weight experts outvoted the highest weight expert.

These statistics provide an upper theoretical bound on the accuracy increase that can be achieved by ideal selection and learning schemes. We captured one of the possible advantages of weighted accumulation over discrete selection.

6. RESULTS

We discuss results with configuration sizes of 8192 and 4096 entries for GShare, 8192 and 4096 entries for Global, and 1028 and 2048 entries for the local branch predictor. The ensemble configurations are labeled with the GShare-Global-Local predictor sizes.

Our ensemble scheme yielded mixed results. First, it was performant where traditional schemes failed. Second, it managed to reduce correctable misprediction rate in some benchmarks which consist of highly uncorrelated branches (figure 9). However it failed to provide a general solution that worked for all benchmarks. There are several reasons that add up to the lower performance of ensemble. We observed high levels of mispredictions where all experts were incorrect (Figure 6), and mispredictions where majority of experts were correct (Figure 7). We had directly adapted the prediction counter updating policy of successful traditional branch prediction schemes. Namely, it is where all predictors are updated with the direction of the outcome of a branch. If taken, a predictor counter will increase. If not taken, a predictor counter will decrease. The number of cases where all of our experts in the ensemble are incorrect shows that this policy is incompatible with an ensemble. For an ensemble to work efficiently, it is required to have the number of mispredictions from individual experts to be low, or at least two or more experts are correct. When two or more experts are correct, the selection/accumulation scheme comes into play. The ideal scheme should be able to allow these correct experts to outvote the highest weighted expert which may be incorrect. The learning and unlearning rates have a high impact on this. However, the weight scheme also needs to balance the weight range with the learning rate in order to adapt the overall ensemble to new sections of the program with new branch patterns. Our statistics show that our weighted accumulation scheme does not live up to exceptations. On average, we observe that 15% of mispredictions had a majority correct despite an incorrect verdict. The highest performing 8K GShare, 8K Global and 2K Local configuration experienced 10% instead.

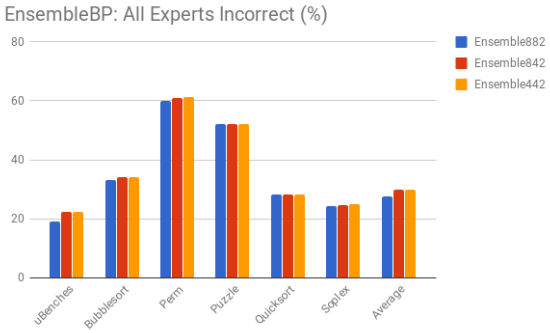

Figure 6: Mispredictions where all experts were incorrect

One of the motivations for creating an ensemble based branch predictor was the number of "correctable" mispredictions. Our weighted accumulation policies were unable to recover some of the "correctable" mispredictions from the baseline discrete selection policy. Our ensemble scheme fell short of its target due to the massive design space exploration that is necessary to construct a well-performing ensemble. In the design space exploration, ideal levels of learning/unlearning rates and the trade-off between adaptation rate and confidence levels must be identified. These rates have to be explored both for the accumulation/selection scheme, and the individual predictors themselves. Adapting traditional schemes is incompatible with the ensemble, as indicated by our results. Further, the group of experts forming the ensemble drive the prediction accuracy for targeted

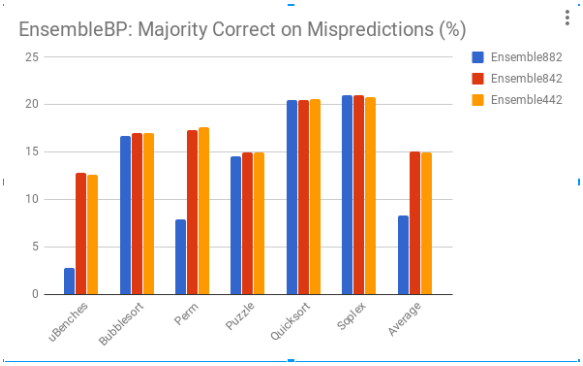

Figure 7: Mispredictions where majority of experts were correct

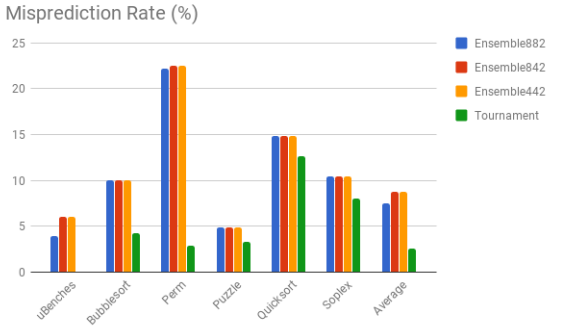

Figure 8: Misprediction rates

applications and hence determining the experts to be used gains significance. Ultimately, we can not conclusively say that a weighted-accumulate ensemble would always perform worse than discrete-selection, as the potential of the ensemble is still shown in our results. Our own scheme however, fell short of determining the correct updating policy and more profiling, data collection and tuning is necessary to evaluate the idea of a learning ensemble overall. Here, we presented a first step on evaluating a hardware ensemble realization in the context of the binary branch prediction problem. We make several design choices that present some benefits and other downsides. We believe further investigation is needed in the aforementioned variables.

7. REFERENCES

[1] S. McFarling, "Combining branch predictors," Tech. Rep. 36, DEC Western Research Laboratory, 250 University Avenue Palo Alto, California 94301, June 1993.

[2] A. N. Eden and T. Mudge, "The yags branch prediction scheme," in *Proceedings. 31st Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 69–77, Nov 1998.

[3] A. Seznec, "The l-tage branch predictor," *The Journal of Instruction-Level Parallelism*, vol. 9, May 2007.

[4] G. H. Loh and D. S. Henry, *Applying Machine Learning for Ensemble Branch Predictors*, pp. 264–274. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[5] V. Vovk, "Universal forecasting strategies," *Information and Computation*, vol. 96, pp. 245–277, 1992.

[6] N. Littlestone and M. Warmuth, "The weighted majority algorithm," *Information and Computation*, vol. 108, no. 2, pp. 212–261, 1994.

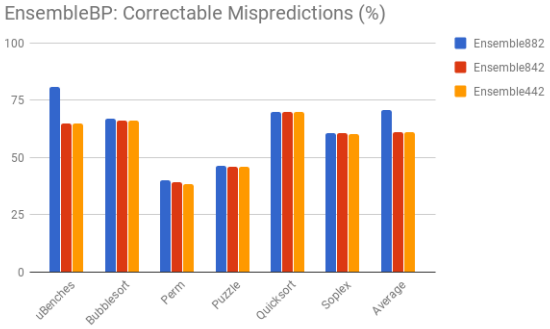

Figure 9: Correctable predictions in the ensemble predictor

[7] N. C. Oza, *Online Ensemble Learning*. PhD thesis, University of California, Berkeley, 2001. AAI3044618.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug. 2011.

[9] C. Lattner and V. Adve, "Llvm: a compilation framework for lifelong program analysis transformation," in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pp. 75–86, March 2004.