209AS1 CA1 Report

*Dual Perceptron Branch Predictor

Fuyao Zhou Jacob Yang

***Abstract*—The goal of this project is to design a new and accurate branch predictor. Your designs will be evaluated against each other to form a championship. This was originally designed and used in the Branch Predictor Championship Competition at ISCA 2016, and all the credits go to the original designers and organizers of the framework.**

***Index Terms*—computer architecture, branch predictor, perceptron, machine learning**

I. Design Methodology

A. Branch History Table

Our approach maintains two branch history tables. One of them keeps track of the long branch history, which covers a large window of the branch history capturing long-period and function call dependency. The other table contains short branch history, which adapts to inner loops and short-period dependency. If we only rely on the long branch history table, local variations will have less impact on the prediction result, while if we only utilize short branch history table, long-period dependency will be ignored, as well.

Besides the branch history tables, we need two more weight matrices. These matrices are 2D 8-bit integer arrays to save memory usage. The size of the tables are empirical values from multiple runs. A clip function is used to prevent weight and bias values from going too large. The weights are used in a simple linear classifier.

$$y = w_0 + \sum_{i=1}^{n} w_i \cdot h_i$$

B. Predict

The predict function is called to make the prediction by directly multiplying the weights with the branch history and accumulating the results. To get the final prediction, we need to compute the weighted sum of the results from the long history perceptron and short history perceptron. In our design, we propose to use an equal ratio of the results. If the combined result is larger than or equal to 0, our predictor makes a taken prediction. If the combined result is negative, a not taken prediction is made.

C. Update

The update function is called after the actual branch instruction is resolved and we would possibly update the branch history table if we make a misprediction or we are not confident enough, which means the absolute value of the combined result is lower than a certain threshold. The threshold value is proportional to the history length and is given by

$$threshold = 1.93 \times \text{HIST\_LEN} + 14$$

The weight matrices and the branch history tables are both updated. The weight matrix is updated by multiplying and accumulating the corresponding history bit with the prediction direction. The history tables are updated by shifting the table to fit one more entry in the front of the table.

II. Design Storage Overhead

There are a total of 4 2D arrays for our predictor. For our configuration, the memory they use is:

$$2^{10} \times 65 \times 8 + 2^{10} \times 9 \times 8 + 64 \times 8 + 8 \times 8 = 75.848\text{KB}$$

All the entries in the tables are stored as type $int8\_t$.

III. Trade-off

A. Result Analysis

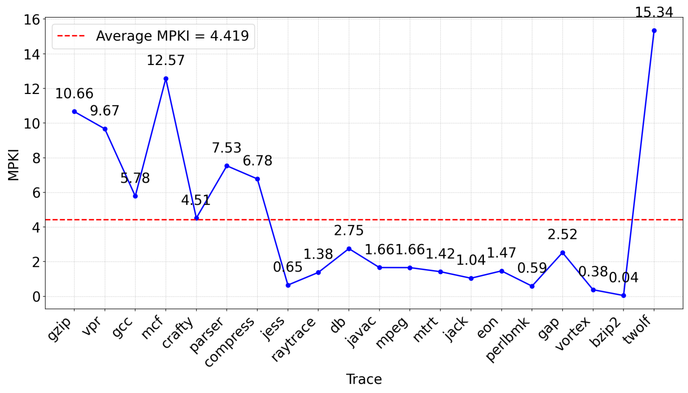

Fig. 1. MPKI results across different benchmarks.

Figure 1 illustrates the MPKI results of the dual-perceptron branch predictor in a series of trace suites. The predictor achieves a low average MPKI of 4.419, which is better than the traditional Gshare predictor (6 MPKI) or the basic single table perceptron (5 MPKI).

In particular, benchmarks such as `bzip2`, `vortex`, and `perlbmk` show extremely low MPKI values (below 1.0), indicating that the predictor is highly effective in capturing short-term and moderately correlated branch patterns. On long-dependency intensive benchmarks like `mcf` and `twolf`, the MPKI values are relatively higher but still got reduction since the ability of the long-history perceptron to capture deep correlations. On short-term loop-heavy benchmarks like `gzip` and `parser`, the short-history perceptron also quickly adapts to local patterns, ensuring fast convergence.

Above all, the dual-perceptron design effectively balances across a variety of branch behaviors.

B. *Trade-off Analysis*

This predictor achieves both good accuracy (4.419 MPKI) and low storage (75.8KB). The 64-bit long history captures distant correlations, while the 8-bit short history adapts to local patterns, achieving a complementary design. The dual-table design with long history slightly exceeds the classical 64KB, but ensures high prediction accuracy. Also, the simple adder + weight clipping is simple and efficient, ensuring a lightweight inference path and low latency. Although the dual table updates slightly increase training energy consumption, the predictor converges quickly, making the overall dynamic energy acceptable.

In conclusion, the predictor strikes an effective balance among long and short history coverage, small storage overhead, low inference complexity, and efficient training, making it suitable for modern high-performance branch prediction frontends.