Branch Prediction Models Exploration

Daniela Trevina
Department of EECS
Texas A&M University-Kingsville
Kingsville, USA
daniela.trevino@students.tamuk.edu

Daniela Lopez
Department of EECS
Texas A&M University-Kingsville
Kingsville, USA
daniela.lopez@students.tamuk.edu

Mengxiang Jiang
Department of EECS
Texas A&M University-Kingsville
Kingsville, USA
mengxiang.jiang@students.tamuk.edu

Samah Allahyani
Department of EECS
Texas A&M University-Kingsville
Kingsville, USA
samah.allahyani@students.tamuk.edu

Ugochukwu Onyeakazi
Department of EECS
Texas A&M University-Kingsville
Kingsville, USA
ugochukwu.onyeakazi@students.tamuk.edu

***Abstract*—**

I. Introduction

Branch prediction is a crucial component in modern computer architecture to improve performance and energy efficiency. When executing a program, the processor encounters branches in the control flow, where it must decide which path to take based on a conditional statement. Incorrect branch predictions result in wasted computation cycles and negatively impact the overall performance of the system. To mitigate this issue, branch predictors are employed to predict the outcome of the conditional statements, and enable the processor to pre-fetch instructions from the predicted path, reducing the overhead of branching.

Over the years, a variety of branch prediction models have been developed to improve the accuracy of predictions. These models range from static branch predictors that use heuristics to predict the outcome of branches, to more sophisticated dynamic branch predictors (all models tested in this paper are dynamic) that rely on past history to make predictions. Furthermore, the emergence of machine learning and artificial intelligence has led to the development of more advanced branch prediction techniques.

The aim of this paper is to explore the various branch prediction models and compare their performances. We will examine the underlying principles of each model, the algorithms used to make predictions and their implementations in a simulated processor. We will also discuss the challenges associated with branch prediction, such as the trade-off between prediction accuracy and hardware complexity.

II. Models

In this section, we will cover the various branch predictor models used in the simulation.

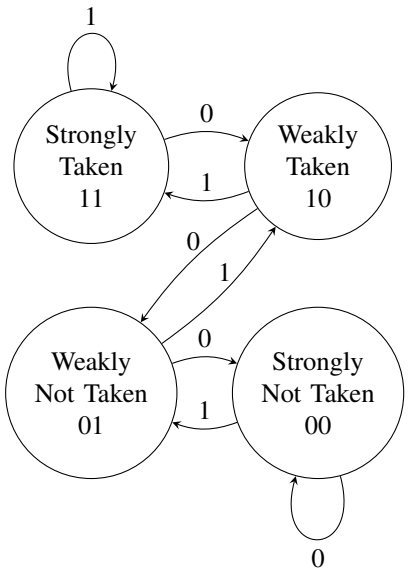

Fig. 1. Bimodal Predictor

A. Bimodal Predictor

This model was first proposed by James E. Smith as an improvement on the single-bit last-time predictor [10]. This is a local history predictor in the sense that the Program Counter (PC) is used to differentiate branches and their past outcomes. It stores two bits in the branch target buffer (BTB) indicating whether this particular branch was taken or not taken in the last two times encountered. These two bits encode a state machine with four states shown in Fig. 1. The *Strongly Taken* and *Weakly Taken* states will predict *taken* for the current branch, while the *Strongly Not Taken* and *Weakly Not Taken* states will predict *not taken*.

B. Gshare Predictor

This model was first proposed by Scott McFarling as an improvement on the global history predictor [4]. A normal

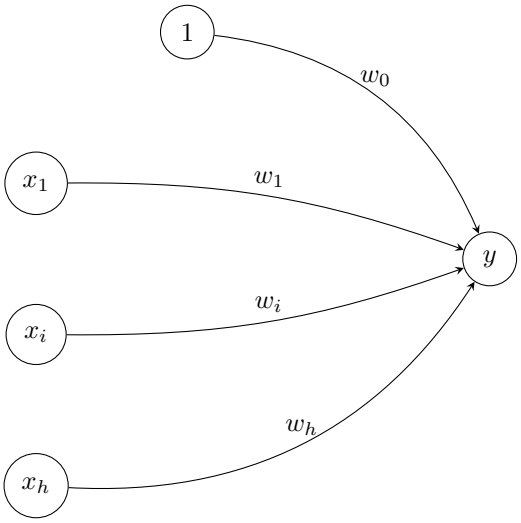

Fig. 2. Perceptron Predictor

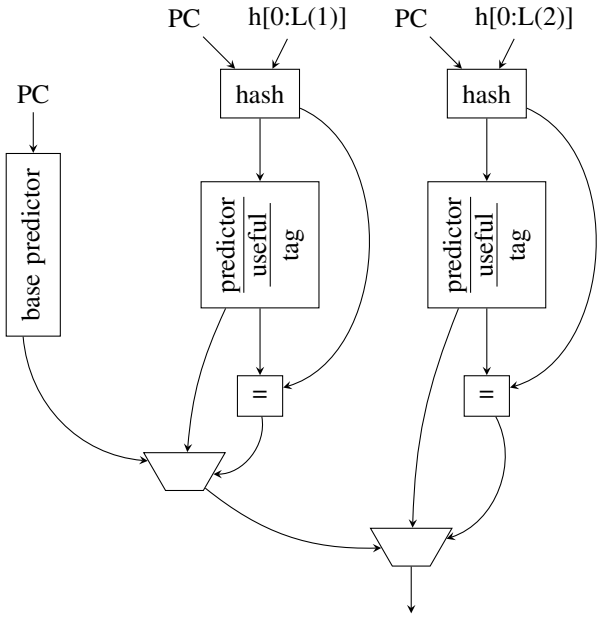

Fig. 3. 3-Component TAGE Predictor

global history predictor uses the single history from the Global History Register (GHR) to index into the Pattern History Table (PHT) in order to predict all branch outcomes regardless of the PC. Gshare XORs the PC with the GHR for indexing into the PHT, which adds some local context information for the predictor to take into account.

C. Perceptron Predictor

This model is a single-layer version of an artificial neural network that can identify and classify patterns, first applied to branch prediction by Daniel A. Jimenéz and Calvin Lin [2]. In the model shown in Fig. 2 are the input vector (x), the weight vector (w), and the output (y). The inputs correspond to values taken from the global history register, with the exception of x_0, the bias, always set to 1. The output's sign determines the prediction. If the sign is negative, the branch is not taken, otherwise it is taken. Equation (1) shows the calculation of the output:

$$y = w_o + \sum_{i=1}^{h} x_i w_i \tag{1}$$

The weights of the perceptron are trained using the algorithm below:

```
if sgn(y) != t or abs(y) <= theta  then
    for i := 0 to n do
        w[i] := w[i] + t*x[i]
    end
end
```

t is 1 if the branch is actually taken or -1 if not. θ is the threshold to determine when training should stop. w_i is incremented if t and x_i agree, and decremented if they disagree.

D. Hashed Perceptron Predictor

This predictor is based off of the perceptron branch predictor from the previous section(II-C) but with a few important changes. The first change is instead of a single table for the weights, multiple independently indexed tables of perceptron weights are used, introduced by Jimenéz [1]. The second is using the hash of the global history rather than just the global history for indexing in order to reduce the number of independent tables, contemporaneously by Seznec [6], Tarjan and Skadron [9], and Loh and Jimenéz [3]. The third is instead of using a fixed global history length, exponentially increasing global history lengths are used for indexing, by Seznec [5] [7]. This idea is actually one of the key parts of the TAGE predictor described in the next section(II-E). The last is to dynamically adjust the θ value for the training, also by Seznec [5] [7].

E. TAGE Predictor

TAGE stands for TAgged GEometric history length branch predictor, and it was introduced by André Seznec [8]. The first key idea, as mentioned earlier, is using exponentially increasing global history lengths for indexing into multiple tables, in this case PHTs while for the hashed perceptron it was the weights. The second is to intelligently allocate the PHT entries to different branches. Fig. 3 shows a 3-component TAGE predictor, while the TAGE predictor evaluated in our simulations has 13 components.

F. L-TAGE Predictor

This predictor is based off of the TAGE predictor from the previous section(II-E), but with the addition of a loop predictor. It was introduced by Seznec and won the 2nd Championship Branch Prediction competition [6]. A loop predictor first predicts the limit on how many iterations the loop will execute. It keeps track of the number of times the loop as executed, and as long as this count is below the limit, it will predict taken.

III. Methodology

The study conducts experiments using the ChampSim Simulator. Each predictor is implemented in the branch prediction unit of the simulator. The predictors are then tested on application benchmarks from Standard Performance Evaluation Corporation (SPEC) suite of benchmarks. The performance of each predictor is recorded and presented in the next section (**??**).

References

[1] Daniel A Jiménez. Fast path-based neural branch prediction. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 243–252, 2003.

[2] Daniel A Jiménez and Calvin Lin. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206. IEEE, 2001.

[3] Gabriel H Loh and Daniel A Jiménez. Reducing the power and complexity of path-based neural branch prediction. In *Proceedings of the 5th Workshop on Complexity Effective Design (WCED5)*, pages 1–8. Citeseer, 2005.

[4] Scott McFarling. Combining branch predictors. Technical report, Citeseer, 1993.

[5] André Seznec. The o-gehl branch predictor. *The 1st JILP Championship Branch Prediction Competition (CBP-1)*, 2004.

[6] André Seznec. Revisiting the perceptron predictor. *PI-1620, IRISA, May*, 2004.

[7] André Seznec. Analysis of the o-geometric history length branch predictor. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 394–405. IEEE, 2005.

[8] André Seznec and Pierre Michaud. A case for (partially) tagged geometric history length branch prediction. *The Journal of Instruction-Level Parallelism*, 8:23, 2006.

[9] Kevin Skadron and David Tarjan. Revisiting the perception predictor again. 2004.

[10] James E. Smith. A study of branch prediction strategies. In *Proceedings of the 8th Annual Symposium on Computer Architecture*, ISCA '81, pages 135–148, Washington, DC, USA, 1981. IEEE Computer Society Press.