Proposal of Computer Arch Project I

I. Introduction of Branch Prediction

In the instruction set of modern CPU, there are lots of basic instructions that functions differently. Through combining these basic instructions together in certain sequences, we can program the CPU to do complex computations and solve real-life problems. However, for complex applications, it's impossible that they can be solved by computing all the way to end, those applications are always involved with some if/loop logics. Thus, except for computation instructions like ADD, MULTIPLY, etc, there are also certain kinds of instructions exist to control the instruction flow and select next instruction to execute on-the-fly. Though branch instructions enable us to integrate more daily logics into computer programs, it might at the meantime break the smooth flow of instruction fetching and execution in highly parallel computer systems [1]. Such problems result in delay since the result of a branch instruction changes the location of instruction fetches and the execution of next instruction must wait until the conditional branch decisions are made. To reduce such delays, branch predictors are introduced in [1] to foresee the next instruction that a branch instruction might point to before it was executed. With branch prediction, we can start fetching, decoding and even executing the next instruction beforehand, which substantially improved the performance of modern pipelined microprocessors.

II. Related Works

Generally, the former researches conducted on Branch Prediction can be divided into two categories, static branch prediction and dynamic branch prediction. Here we present a brief review of predictors of those two types.

A. *Static Branch Prediction*

Static branch prediction is the earliest and simplest branch prediction strategy. As the name itself shows, such predictors produce static results, i.e., they made prediction decisions before program execution. Classic static branch prediction strategies include but not limited to:

- Predict all branches will be taken.
- Predict that all branches with certain operation code will be taken; predict that the others will not be taken.
- Predict all backward branches (toward lower addresses) will be taken; predict all forward branches will not be taken.

Despite their simplicity, the actual performance of those strategies is considerably good, the result accuracy ranges from 60% to more than 90% depending on different benchmarks used [1]. Afterwards, more static branch predictors are proposed, one typical example among them is static branch predictor with neural networks. Using this approach, the branch direction is predicted in compile-time by program features, its prediction results reached an 80% correct prediction rate [2].

B. *Dynamic Branch Prediction*

Dynamic branch prediction makes prediction decisions based on history branch taken data. Since dynamic predictors can be self-tuning in the runtime, they can provide better performance most of the time. Typical dynamic predictors include:

- Branch Target Buffer [7]: use 2-bit saturating up-down counters to collect history information for predictions.
- Two-level adaptive training branch prediction [8]: the most widely-used strategy that has been put in many real computer architecture.

- Dynamic Perceptron Branch Prediction(DPBP) [5]: first successful integration of neural networks into dynamic branch prediction.
- Path-Based Neural Branch Prediction(PBNBP) [6] and Piecewise Linear Branch Prediction(PLBP) [3]: improved versions based on perceptron approach.
- L-TAGE [9]: state-of-the-art branch prediction with high accuracy.

III. Fast Path-Based Neural Branch Prediction

After careful evaluation of other branch prediction methods, I finally decided to choose PBNBP as my target. In the next two sections, I will state the reason of choosing this approach and present a review of its design.

A. Reason

Reviewed the evolution of branch predictors, I first decided to implement a dynamic predictor instead of static ones, the reasons shall be three fold:

- Static prediction strategies in general perform worse than dynamic techniques.
- Static predictors are unable to tune themselves on the fly, while the prediction strategies always vary greatly between different programs.
- The program logic of static predictors is often simple and unchallenging.

When choosing dynamic branch predictors, I first narrowed down several high-quality research papers, which are perceptron, PBNBP, piecewise linear branch prediction and L-TAGE, among which perceptron is the earliest, L-TAGE performs the best and the other two is improved strategies based on or inspired by perceptron. After consulting Prof. Jiménez, I choose Path-Based Neural Branch Prediction approach for following reasons:

- The paper of L-TAGE is too vague to understand, making it hard to implement in a short time.
- Perceptron as the precursor of dynamic neural network branch prediction is too simple and its performance might not help me win the prize.
- The size of a version of PLBP is 65,789, which exceeds our project requirements. [3]
- PBNBP approach has a considerably good accuracy and enjoys a low latency, while occupying a reasonable space.

B. Design

Before introducing the design of PBNBP, we first review the concept of perceptron learning [4]. The perceptron is a binary classifier based on supervised machine learning, it can classify a series of vector input (such as $< \mathrm{x}_1, \mathrm{x}_2, \dots, \mathrm{x}_\mathrm{n} >$) into two categories, for example, taken or not taken. To adapt it to our branch prediction application, we use x_i in those vectors to represent the bits of a global branch history shift register. The mechanism of a perceptron is shown in Fig.1, and the output of a perceptron is computed as:

$$y = \omega_0 + \sum_{i=1}^{n} x_i \omega_i$$

Similar to Perceptron Branch Predictor(PBP), PBNBP is also based on perceptron, the simplest neural networks approach. Since dynamic branch predictors need to make decisions based on branch taken history, PBNBP keeps a global history shift register that records the outcomes of branches as they are executed or speculatively as they are predicted. In order to do this, it keeps an $\mathrm{n} \times (\mathrm{h} + 1)$ matrix $\mathrm{W}_{\{\mathrm{n}\times(\mathrm{h}+1)\}}$, in which every element is an 8-bit bytes integer weights,n is a design parameter. Each row of the matrix is an $(\mathrm{h} + 1)$-length weights vector, each vector stores the weights of one perceptron. The first weight elements of any row is known as the *biased weight*. The Boolean vector $\mathrm{G}[1..\mathrm{h}] \in \{1..\mathrm{h}\} \times \{\mathrm{taken}, \mathrm{not\ taken}\}$ represents the global history shift register [5].

Different from PBP, only the bias weight in PBNBP is used to predict the current branch. The bias weight is added to a taken total of last h branches, with each summand added during the processing of a previous branch [6].

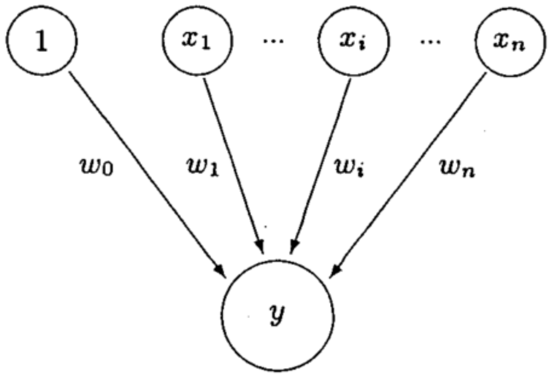

Fig.1 Perceptron Mechanism

The history length of our predictor is set as $h = 7$, thus the set of weight used in predicting branch b_t is $x[0\ldots7]$. To take the path towards our current branch into consideration, $x[0..7]$ weights are built by accessing different positions in the weights vectors associated with the path from $b_{\{i-7\}}$ to b_t. At the meantime, we record a running total count flag which keeps accumulating y_{out}. The detailed prediction and update algorithm for the perceptron is temporarily omitted here for brevity and will be discussed later.

C. Timetable

As a plan to follow up in the development of PBNBP branch predictor, I carried out a time table as follows:

TABLE I. TIME TABLE

Task	Planned Finish Time	Current Status
Literature Review	09/20/2016	Finished
Environment Investigation	09/27/2016	Ongoing
PBNBP Core Implementation	10/10/2016	Ongoing
PBNBP Integration	10/20/2016	TODO
Test and Document	10/25/2016	TODO

REFERENCES

[1] Smith J E. A study of branch prediction strategies[C]//Proceedings of the 8th annual symposium on Computer Architecture. IEEE Computer Society Press, 1981: 135-148.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68-73.

[2] Calder B, Grunwald D, Jones M, et al. Evidence-based static branch prediction using machine learning[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1997, 19(1): 188-222.K. Elissa, "Title of paper if known," unpublished.

[3] Jiménez D. Idealized piecewise linear branch prediction[J]. Journal of Instruction-Level Parallelism, 2005, 7: 1-11.

[4] Block H D. The perceptron: A model for brain functioning. i[J]. Reviews of Modern Physics, 1962, 34(1): 123.

[5] Jiménez D A, Lin C. Dynamic branch prediction with perceptrons[C]//High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on. IEEE, 2001: 197-206.

[6] Jiménez D A. Fast path-based neural branch prediction[C]//Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on. IEEE, 2003: 243-252.

[7] Lee J K F, Smith A J. Branch prediction strategies and branch target buffer design[C]//Instruction-level parallel processors. IEEE Computer Society Press, 1995: 83-99.

[8] Yeh T Y, Patt Y N. Two-level adaptive training branch prediction[C]//Proceedings of the 24th annual international symposium on Microarchitecture. ACM, 1991: 51-61.

[9] Seznec A. The L-TAGE branch predictor[C]//Journal of Instruction Level Parallelism. 2006.