Using Machine Learning for the Improvement of Branch Prediction

Zimu Zheng, Wentao Li, and Yunhang Zhu, *IEEE*

Abstract— This paper studies the dynamic branch prediction to finding out how to use machine learning to improve the branch prediction. Normally the accuracy of the branch prediction is the most important part. This is because more correct branch of the processor takes, the faster the performance is. The traditional branch predictor without the neural network can only give a low accuracy. So this paper analyzes the disadvantages of the normal branch prediction and tries to use the machine learning to make it better. The paper design and analyzes two artificial neural network---Learning Vector Quantization (LVQ) and Perceptron algorithms using in the dynamic branch prediction. These two algorithms can improve the accuracy of the dynamic branch prediction and improve the performance of instructions.

Index Terms—Machine Learning, Branch Prediction, Learning Vector Quantization, Perceptron

1 Introduction

Branch prediction is a method used by modern processors to improve the speed of CPU execution. It predicts the branch process of the program and then reads the instruction and decoding of one of the branches in advance to reduce the time of the waiting decoder.

There are two kinds of branch prediction methods: static prediction and dynamic prediction: static prediction method is simple, such as prediction never transfer, JMP, forecast backward transfer and so on. It doesn't work according to the conditions of execution and historical information, so the accuracy of this kind of branch prediction is not very good. Dynamic prediction, on the other hand, predicts the future based on the transfer status history of the same transfer instruction.

As the conditional branch in the program is executed according to the result of the instruction in a program after the pipeline is processed, the front stage circuit of the pipeline will be idle when the CPU is waiting for the result of the instruction. Thus the waste of the clock cycle is inevitable. If the CPU can predict whether the branch is taken or not before the result of the preceding instruction comes out, it can decode and execute the corresponding instructions in advance so that the idle wait for the pipeline can be avoided, and the total speed of the CPU will be improved. But on the other hand, once the preceding instruction results are proved to be wrong, that is, the wrong branch prediction is produced, then all the instructions and results that have been loaded into the pipeline must be cleared and then reloaded with the correct instruction. This may take more time than not using branch prediction just waiting for the result to execute the new instruction.

Therefore, the error of the branch prediction does not lead to the error of the result, but only leads to the halt of the pipeline. If there is a prediction which can have better accuracy, the branch prediction can improve the performance of the pipeline.

As the instructions used in processors keep increasing, the accuracy of the dynamic branch prediction becomes more and more important. Therefore, even if a few percents of the wrong branch prediction will have considerable performance loss. If the branch prediction wants to improve the performance, the branch must be detected in the dynamic instruction stream and the direction required for each branch and the target address must be correctly predicted. In addition, all of these must be completed in time to get instructions from the branch destination address without interrupting the flow of new instructions to the processor pipeline.

2 Why Traditional Branch Prediction Needs Improvement

2.1 Static Branch Prediction

Static prediction is the simplest branch prediction technique because it does not depend on the dynamic historical information of code execution. Instead, it depends only on the branch instruction itself.

The earliest implementation of SPARC and MIPS uses single directional static branch prediction: always predicting conditional jump that does not occur, so it is always sequential to take an instruction to conjecture execution. When the conditional jump instruction is evaluated, the non-sequential code address will be loaded. The two kinds of CPU evaluate branch instructions in the decoding phase takes 1 cycles. Therefore, the branch target needs two cycles to fetch. The two processors will insert a branch delay gap when the branch instruction enters the execution phase of the pipeline. When the branch instruction completes the execution of the pipeline, it has been able to determine whether to jump or not, when it is possible to decide whether the subsequent order will be continued or the new instruction that is skipping into the pipeline.

More complex static prediction assumes that backward branches will occur, and forward branches will not

occur. Backward branching means that the new address to jump is lower than the current address. This helps to match the recurrent control structure of the program.

However, the static branch prediction only depends on the branch code or part of the code to predict. The accuracy of the static branch prediction normally is about 55%~60% which is pretty low for branch prediction The performance of static branch prediction is limited by the fact that it can not adapt to branch bias dynamically according to the execution history of branch instructions.

2.2 Dynamic Branch Prediction

Dynamic branch prediction is a technique that has been developed by modern processors. The simplest dynamic branch prediction strategy is the Branch Prediction Buff or the branch history table.

Branch History Table (BHT), as the name implies, is a table of the record of branch history information which used to determine whether a branch instruction is taken or not. The record is about the jump information. For example, people can use 1bit records, such as 1 means jump and 0 means not jump and the index of this table is the PC value. Considered a 32-bit system, if people want to record the complete 32-bit branch history, they will need 4Gbit of memory, which is totally beyond the hardware support capabilities provided by the system. So generally they use the last 12 bits of the instruction as the index of the BHT form so that with only a 4K bits form, the whole branch history can be recorded. Obviously, with the efforts and analysis of branch prediction, people found that recording a branch with 1 bit in the BHT is not accurate enough. Recording with 2 bits, however, is pretty good, and recording with 3 bits or more seems to be the same to 2 bit. So in the BHT, generally use 2bit to record to determine the jumps of a branch: for example, 11 and 10 indicate that this branch will jump; 01 and 00 indicate that the branch will not jump. This 2-bit counter is so-called a saturation counter.

BTB - using to record the branch address of a branch instruction. Because of the address of the instruction which is stored, such as 32-bit address, this form cannot store as much as the BHT content. If it supports 4K instructions, it will need 128Kbit of storage space, which can almost catch up with the capacity of an L1 Cache. So the BTB is generally small, 32 items or 64 items. Since the capacity of the BTB is small and it is used to record the branch instruction's jump address if this instruction does not jump, its next instruction will be PC+4 and it will not be recorded in the BTB.

However the traditional 2bit dynamic branch prediction can only reach an accuracy of 90% because they only use 4 situations to predict the branch: Strongly not taken, weakly not taken, weakly taken, and strongly taken. But sometimes the branch doesn't work in the predicted way and people need to find a better method to let the processor to predict the branch not only based on the history table that already is set up but also keeps changing the table. So this paper analyzes and designs two algorithms using machine learning method to improve the dynamic branch prediction.

2.3 Machine Learning

Machine learning is a branch of artificial intelligence. Machine learning has developed into a multidisciplinary interdisciplinary subject over the past 30 years, involving many disciplines such as probability theory, statistics, approximation theory, convex analysis, and computational complexity theory. Machine learning theory is mainly to design and analyze some algorithms that allow the computer to automatically "learn". Machine learning algorithm is a kind of algorithm for automatically analyzing and obtaining unknown data from data because learning algorithms involve a large number of statistical theories, machine learning and inference are closely related to statistics, also known as a statistical learning theory. In terms of algorithm design, machine learning theory focuses on effective learning algorithms that can be implemented. Many inference problems are difficult to follow, so some machine learning research is an approximate algorithm which is easy to handle.

Machine learning has been widely used in data mining, computer vision, Natural Language Processing, biometrics, search engines, medical diagnostics, detection credit card fraud, securities market analysis, DNA sequence sequencing, voice and handwriting recognition, strategic games and robots.

2.4 Artificial Neural Network

In the field of machine learning and cognitive science, an artificial neural network is a mathematical model or calculation model that imitates the structure and function of the neural network (the central nervous system of animals, especially the brain), which is used to estimate or approximate the function. The neural network is calculated by a large number of artificial neuron connections. In most cases, the artificial neural network can change the internal structure on the basis of external information, and it is an adaptive system. A modern neural network is a nonlinear statistical data modeling tool.

The idea of constructing a neural network is inspired by the operation of biological neural networks. Artificial neural networks are usually optimized by a learning method based on Mathematical Statistics. So artificial neural networks are also a practical application of mathematical statistics. By the standard mathematical method of statistics, we can get a large number of local structural spaces that can be expressed by functions. In the field of artificial intelligence. On the other hand, in the field of artificial intelligence, we can make decisions in artificial perception through the application of mathematical statistics. This method is better than formal logic reasoning.

Like other machine learning methods, neural networks have been used to solve various problems, such as machine vision and speech recognition. These prob-

lems are difficult to be solved by traditional rule-based programming.

3 Machine Learning to Improve Branch Prediction

This paper uses two methods of machine learning, LVQ and Perceptron, to improve the accuracy of the branch prediction. Each method contains an algorithm for the changing table.

3.1 LVQ

3.1.1 Why use LVQ

LVQ is a special type of an artificial neural network, more specifically, LVQ applies a winner-take-all Hebbian learning-based approach. LVQ is a supervised learning algorithm. The strength of it is that it only has two neuron layer and only use the distance to imitate the model. So it will be much faster than usual neuron network algorithm such as backpropagation and forward algorithm. The main idea that we choose LVQ is to design a list of prototype vector to compare with the branch actual result and update it after each comparison. While the comparison result will be used as a prediction result.

3.1.2 Design of LVQ

The key part of the update is:

$$q_{correct} = q_i + \eta \bullet (x_i - q_i) \quad (1)$$

$$q_{wrong} = q_i - \eta \bullet (x_i - q_i) \quad (2)$$

When the prediction is correct, using (1) to update.

When the prediction is wrong, using (2) to update.

The key part of the comparison is the Hamming distance of x and q. For the local history, the paper uses a dictionary to realize that the local history table where the key is the address of the branch and item is a 4 bits length recent actual branch result of this address. For the global history, the paper uses a 12 bits length recent branch predict recent branch result. The input vector to the LVQ algorithm is a combination of address, local history and global history in bits. The training part of this algorithm is keeping updating the prototype vector using (1) and (2) which is the learning vector in machine learning. The branch prediction predicts according to the hamming distance calculated from (1) and (2) to the input vector.

3.1.3 Running Steps of LVQ

Step 1: Initialize the taken and not taken prototype vector.

Step 2: Generate the input vector according to the trace address, local history table, and global history table.

Step 3: Predict the branch according to the distance from prototype

Step 4: Train the taken and not taken prototype vector and update the local and global history table.

3.2 Perceptron

3.2.1 Why use Perceptron

Perceptron is an algorithm for supervised learning of binary classifiers. And it is a linear classifier which only does a weight update in k times operation and k is the length of global history table. Thus, it is very suitable to solve our branch prediction problem, accurate and fast.

The main idea of the perceptron is to initialize an address based list of perceptron model and for each model we use the data to test if it is larger than 0. If not, train it. And if the sum of the test is larger than 0, we predict taken, vice versa.

3.2.2 Design of Perceptron

The model form of perceptron algorithm is:

$$f(x) = sign(w*x + b) \quad (3)$$

If the sum of f(x) larger than 0, taken, vice versa.

In this paper, the perceptron table is that there are n perceptrons which are based on the number of the address. However, the number of address is so large that has to be hashed to map the address in 1000 groups. In the meanwhile, the paper uses (3) and makes x to be the global history table. And then use a random number to initialize the perceptrons.

According to the definition of the perceptron, if the $f(x)*y_i$ is less than 0 for each trace, the f(x) model will be updated. Also if f(x) is small enough, it needs to be updated too. The paper uses the perceptron model to generate the f(x) result as x is the global history.

3.2.3 Running Steps of Perceptron

Step 1: Initialize the 80bit global history table and the perceptron table.

Step 2: Calculate the iteration of perceptron model.

Step 3: Keep training the model with the prediction result and updating the model.

4 Simulation Result

This paper runs 6 traces in different 2bit and machine learning methods to test the ability of the dynamic branch prediction. The accuracy of the branch prediction is what people most concern about. Also, the paper shows the speed of different branch prediction which shows how fast the performance is.

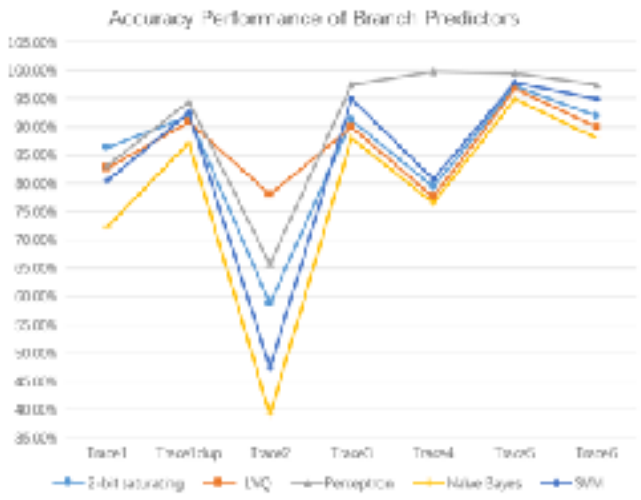

Fig. 1. The simulation of the accuracy of different branch prediction (p.s. Trace1, Trace1dup and Trace2 are around 1000 lines while Trace3 to Trace 6 are about 100000 branch prediction lines)

From this figure we can find that LVQ is steady in the 1000 lines level branch prediction. At the same time, Perceptron has a excellent performance around 95-99% accuracy in trace3 to trace6. It has large improvement in trace4 comparing to other predictor which reflects it's worth.

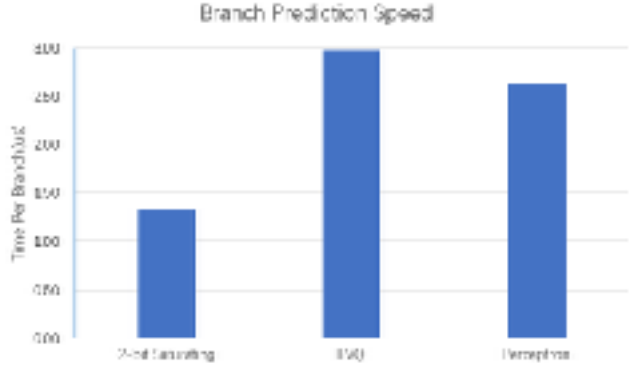

Fig. 2. The simulation based on the time consuming per branch on different branch prediction.

As we can see, the predictors with machine learning algorithm are one time slower than 2-bit saturating branch predictor. However, considering its high accuracy, 'us' level speed and steady performance, we believe it is still agreeable.

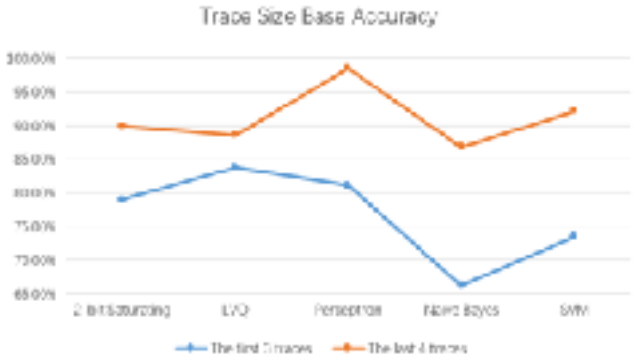

Fig. 3. The simulation of performance speed on different branch prediction.

As we said before, at the first three traces(1000 lines level), LVQ and 2-bit saturating predictors have a good performance and 2-bit branch predictor has the advantage of speed. For the last 4 traces, Perceptron has the best performance. (p.s.the simulation of Naïve Bayes and SVM algorithms are based on previous work.)

Generally speaking, as a perceptron model-based algorithm, the more the branch lines has, the higher its accuracy is. And it does not need to design a local history table. Actually, the perceptron table is its own local history table. As shown in Fig. 2. the only problem of the perceptron algorithm is the speed, so if the branch data is repeated with lower variance in t/nt, it is better to use the 2-bit saturating counter.

5 Conclusion

The paper design two methods, the LVQ and Perceptron, based on machine learning to improve the accuracy of the dynamic branch prediction. As a result, which shown in the simulation, both two methods have achieved the purpose of the design.

The improvement of LVQ branch predictor is the stability comparing with 2-bit saturating counter we made before and other predictors such as perceptron. It is due to its well design local history table and global history table which improve its robustness.

At the same time, the perceptron predictor has a really high accuracy in large scale branch prediction. The perceptron predictor is based on reinforcement learning which can construct a predict model itself and is very suitable to specific branch lines with a startup training data.

Furthermore, since our trace file is different, the paper believes that it is very hard to compare our simulation result with the previous paper result. And the paper believes that in the actual work of architecture, people should combine several predictors together and do a tradeoff between the speed and accuracy.

REFERENCES

1. A. S. Fong and C. Y. Ho, "Global/Local Hashed Perceptron Branch Prediction," *Fifth International Conference on Information Technology: New Generations (itng 2008)*, Las Vegas, NV, 2008, pp. 247-252. doi: 10.1109/ITNG.2008.258
2. B. Kalla, N. Santhi, A. H. A. Badawy, G. Chennupati and S. Eidenbenz, "Probabilistic Monte Carlo simulations for static branch prediction," *2017 IEEE 36th International Performance Computing and Communications Conference (IPCCC)*, San Diego, CA, 2017, pp. 1-4. doi: 10.1109/PCCC.2017.8280494
3. C. Y. Ho, K. F. Chong, C. H. Yau and A. S. S. Fong, "A Study of Dynamic Branch Predictors: Counter versus Perceptron," *Information Technology, 2007. ITNG '07. Fourth International Conference on*, Las Vegas, NV, 2007, pp. 528-536. doi: 10.1109/ITNG.2007.22
4. C. Zhou, L. Huang, Z. Li and Q. Dou, "SimpleBP: A Lightweight Branch Prediction Simulator for Effective Design Exploration," *2017 International Conference on Networking, Architecture, and Storage (NAS)*, Shenzhen, 2017, pp. 1-2. doi: 10.1109/NAS.2017.8026877
5. D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, Monterrey, 2001, pp. 197-206. doi: 10.1109/HPCA.2001.903263
6. G. Marwa, B. Mohamed, C. Najoua and B. M. Hédi, "Generic neural architecture for LVQ artificial neural networks," *2017 International Conference on Engineering & MIS (ICEMIS)*, Monastir, 2017, pp. 1-7. doi: 10.1109/ICEMIS.2017.8272996
7. G. Steven, R. Anguera, C. Egan, F. Steven and L. Vintan, "Dynamic branch prediction using neural networks," *Proceedings Euromicro Symposium on Digital Systems Design*, Warsaw, 2001, pp. 178-185. doi: 10.1109/DSD.2001.952279
8. J. Chen and L. K. John, "Autocorrelation analysis: A new and improved method for branch predictability characterization," *2011 IEEE International Symposium on Workload Characterization (IISWC)*, Austin, TX, 2011, pp. 194-203. doi: 10.1109/IISWC.2011.6114179
9. J. Rahman, M. N. I. Mondal, M. K. B. Islam, M. A. M. Hasan and S. M. S. Amin, "Gram-positive bacterial protein subcellular localization prediction using features fusion strategy," *2016 9th International Conference on Electrical and Computer Engineering (ICECE)*, Dhaka, 2016, pp. 291-294. doi: 10.1109/ICECE.2016.7853913
10. Jeremy Singer, Gavin Brown, and Ian Watson, "Branch Prediction with Bayesian Networks," 2007 - researchgate.net
11. L. V. M. Ribas and R. A. L. Goncalves, "Evaluating branch prediction using two-level perceptron table," *14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06)*, 2006, pp. 4 pp.-. doi: 10.1109/PDP.2006.34
12. V. P. Bharadwaj and M. Kohalli, "Dual decode architecture for dynamic branch prediction," *2017 2nd International Conference for Convergence in Technology (I2CT)*, Mumbai, 2017, pp. 1140-1143.
    doi: 10.1109/I2CT.2017.8226306
13. Y. Lee and G. Lee, "Detecting Code Reuse Attacks with Branch Prediction," *in Computer, vol. 51, no. 4*, pp. 40-47, April 2018. doi: 10.1109/MC.2018.2141035 W.-K.
14. Yang Lu, Yi Liu and He Wang, "A study of perceptron based branch prediction on Simplescalar platform," *2011 IEEE International Conference on Computer Science and Automation Engineering*, Shanghai, 2011, pp. 591-595. doi: 10.1109/CSAE.2011.5952918