Branch Prediction by Checking Loop Terminal Conditions

S. Patil, N. B.Anne, U. Thirunavukkarasu, E. E. Regentova
Department of Electrical and Computer Engineering
University of Nevada, Las Vegas
Las Vegas, Nevada-89154-4026
{shrutí, naveenba, utthaman, regent}@egr.unlv.edu

Abstract

Branch prediction accuracy is a crucial parameter in determining the amount of parallelism that can be exploited. Current advanced branch prediction techniques are able to attain correct predictions in most cases. Conventionally, the profiling information is used to predict the behavior of a branch. However, they are unsuccessful in predicting specific conditions like branch loop terminations, arrays e.t.c. This hinders them from achieving full accuracy. Neural Networks have an inherent capability to analyze inputs to form meaningful relationships among them. This property of neural networks makes them useful to identify program elements that constitute a branch. In this paper, we investigate further and propose an algorithm to predict correctly specific loop conditions using neural networks which can increase the efficiency beyond the rates reported by existing schemes.

Keywords: Branch prediction. Neural network

1. Introduction

Correct branch prediction is important especially in the age of superpipelined superscalar processors. Highly accurate prediction is required to reduce the number of penalty cycles in the multiple instruction issue processors. The number of penalty cycles decides the performance loss for processors whose pipeline depth and instruction issue rate are high. Branch prediction is an essential part of modern micro architectures. Rather than stalling when a branch is encountered, a super pipelined processor uses branch prediction to speculatively fetch and execute instructions along the predicted path. Machine learning techniques offer the possibility of further improving the performance by increasing prediction accuracy.

In our approach, we consider branch prediction as a specific problem belonging to the pattern recognition and in particular to the use of perceptron based neural networks to predict the branches. The perceptrons have several attractive properties. They are simpler to implement and tune, their training is faster, and they are computationally inexpensive.

In this paper, we investigate the behavior of the loops. We capture the various characteristics of the loops and feed them to a network of perceptrons to correctly predict the branches in the loop. This paper is organized as follows: Section 2 reviews briefly the literature on branch prediction schemes. The proposed technique is described in Section 3. In Section 4, the hardware requirements for the algorithm are sketched and the results are shown in section 5. Section 6 draws the conclusions and plans the future work.

2. Background

Branch prediction performance issues have been studied extensively. J. Smith [1] gave a survey of early simple static and dynamic schemes. The best scheme in his paper is the one which uses 2-bit saturating up/down counters to collect history information which is then used to make predictions. It achieves a prediction accuracy of 99.4% for the 'ADVAN' test bench program. This is perhaps the most well-known technique. McFarling [2] referred to it as *bimodal* branch prediction. It was also referred to as *one-level* branch prediction in Yeh and Patt's paper [3]. Lee and Smith [4] evaluated several branch prediction schemes. They found that the miss rates for opcode prediction ranged from 20.2% to 44.8% with an average of 30.1%. In addition, they addressed how to use branch target buffers to reduce the delay due to target address calculation. McFarling and Hennessy [5] compared various hardware and software approaches to reducing branch cost including using profiling information. Fisher and Freudenberger [6] studied the stability of profile information across separate runs of a program. In many programs with intensive control flow, very often the direction of a branch is affected by behavior of other branches. By observing this fact, Pan, So, and Rahmeh [7] and Yeh and Patt [3] independently proposed correlated branch prediction schemes, also called two-level adaptive branch prediction schemes in Yeh and Patt's paper. Correlation schemes use both single conditional branch history and global branch history. Pan, So, Rahmeh [7] described how both global history and branch address information can be used in one predictor. This new approach achieved a maximum prediction accuracy of about 96.5%. There are several variations of this kind of dynamic schemes by using different indexing method and buffer organizations. Yeh and Patt [3] gave a comparison of these approaches achieving a correct prediction accuracy as of 97%. In designing the 2-bit counter used in many of the dynamic schemes, several variations exist. Yeh and Patt [8] discussed these variations while achieving a best prediction accuracy of 97.2%. McFarling [2] exploited the possibility of combining branch predictors to achieve higher prediction accuracy of 98.1%. He also presented a sharing index scheme, referred to as *gshare*, and a new scheme using combined predictors. Ball and Larus [9] described several techniques for guessing the most common branches directions at compile time using static information. They obtained an average misprediction of 12%$\pm$ 10% for loop branches and 26% for non loop branches. Young and M. Smith [13] [12] introduced the notion of static correlation branch prediction (SCBP) which yields up to a 14.7% improvement in prediction accuracy over static profile-based prediction. In a recent paper [14], Gloy, et al addressed performance issues of this approach. They claimed a better performance in comparison to some dynamic approaches. Several studies [10] [11] have looked at the implications of branches on available instruction level parallelism (ILP). In [15], Gordon and his team explored the suitability of two neural networks, a Learning Vector Quantization Network (LVQ) and a Back Propagation Network, for branch prediction. They considered the performance of a simple LVQ neural predictor and then compared the performance of conventional two level adaptive predictors with a neural predictor using backpropagation network. The LVQ and Backpropagation schemes achieve a misprediction rate of 10.78% and 8.77%. In [16], Jimenez and Lin introduced a perceptron based dynamic branch prediction which has a misprediction rate of 5.35%. Among all the available schemes, the two level adaptive branch prediction scheme [3] achieves a minimum misprediction rate of 3%. The study of the results show that on average 4-5% of branch misprediction rates can be achieved and that the targeted values are of 2-3%.

3. Branch Prediction Algorithm

Small improvements in accuracy can have a large impact on performance; decreasing the misprediction rate from, say, 5% to 4% can decrease the execution time of a typical program by as much as 14%. Here, we propose a novel branch prediction scheme using perceptrons which can achieve a very high accuracy. We assume that all predictions are being made in parallel to the instruction decode (ID) stage of the processor pipeline. The actual prediction is generated by a network of perceptrons. The 80386 instruction set is considered in our study and design.

3.1 Loop elements

A loop is governed by the following elements:
1. The loop terminating condition e.g. Jump if Equal, Jump if Zero, Jump if less than or equal
2. The comparison variables: These are the variables that are compared to determine the terminating condition. These are referred to as *determs* henceforth, in this paper.
3. Condition statements: These statements are used to establish the condition.
4. Determ changers: These are the statements that change the value of determ(s) by one or more *numerical factors*. The changed value(s) is (are) compared in every iteration to determine if terminating condition is satisfied.
5. Loop statements: These are the statements that are included in the loop, but are not determ changers.

```
4004: MOV R4, R3
4005: MOV R1, 10
4006: MOV R2, 2
4007: ADD R4, R3
4008: INC R2
4009: CMP R1, R2
400A: JNE 4007
400B: STO R4, 5000H
```

Fig.1. Assembly code for a basic code that computes R4 $\leftarrow$ R3*10

Consider the code in Figure1. The loop terminating condition is JNE (Jump if NOT Equal), determs are R1 and R2, comparison statement is CMP R1, R2, determ changer is INC R2, loop statements are statements #4007, 4009 and 400A.
A loop terminating condition depends on the result of the compare instruction, and hence on the current values of determs. The terminal value of R2 for the code in Figure 1 is 10. When R2 becomes equal to 10, the iterations are terminated and the sequential statement is fetched. Comparison statements can be a simple compare instruction that compares values of two variables, or it can be an arithmetic instruction that changes the state of Machine Status Word, e.g.: SUB R2,R2, or it can be a statement that compares the value of a variable with an immediate value. The determs can be both registers, or memory locations or an immediate value. Also, determ changers can be one or more statements.
Clearly, analysis is required for determining various loop parameters and to identify the point at which loop would terminate. We research the use of neural networks for this task in this paper. A neural network is flexible for changing loop conditions and their analysis. The neural network is required for identification of branch instruction, loop parameters and determination of terminal values. When this is accomplished, the branch

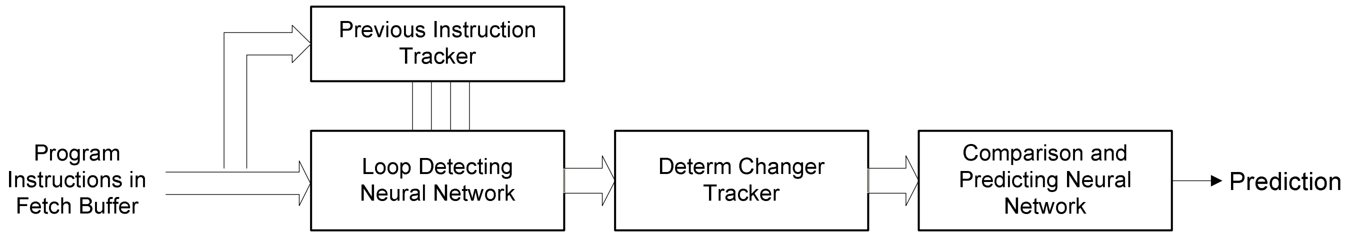
Previous Instruction Tracker
Program Instructions in Fetch Buffer
Loop Detecting Neural Network
Determ Changer Tracker
Comparison and Predicting Neural Network
Prediction

Fig.2. The block diagram of the algorithm

predictor sniffs the values on the data bus to capture the current values of determs.

3.2 Analysis

An analysis scheme is used after the Instruction Fetch stage to determine loop parameters. During the analysis, branch opcodes corresponding to backward jump are identified, and the previous instruction is scanned for variables i.e. determs that change the Machine Status Word. The loop is iterated once to realize the function that changes values of determs. This information is stored with the branch instruction address. Current value of determ(s) and final values are checked. The current value is changed according to the function by the neural network each time the branch statement is fetched. If the final value is also dynamic, then it too is changed similarly. For making a prediction, the perceptron network has to compare the current value and terminal value, and output a "The loop will terminate, do not take the branch" prediction when the values become equal. A neural network is capable of performing this analysis quickly.
This analysis is done in the time period between two subsequent instruction fetch cycles. This way, the output of predictor is ready to fetch the next instruction after a branch instruction is fetched. The hardware proposed as an example of the algorithm in Section 4.1 has a delay smaller than the machine cycle time of the processor used for the example. Input to neural network are the contents of instruction fetch buffer. When contents of IF buffer change, the network starts operation. A small amount of memory is required for storing values of loop parameters for loops. It is not required to store a large amount of history, and once a loop is terminated, its information can be erased to save memory. It can be recomputed within a single iteration if required.
For function calls, when the 'CALL' instruction is encountered, the neural network is simply programmed to predict the end of function call when it encounters a return statement, i.e. 'RET'. Hence, an unconditional call can be predicted.
The block diagram for the above analysis is shown in Figure 2. The four blocks perform loop analysis, computation and comparison and the comparison block outputs prediction, respectively.

4. Hardware requirements of the algorithm

In addition to analyzing the neural network, a small amount of memory is required to process a branch and generate the necessary control signals as shown in Figure 3. Figures 4 and 5 show an elementary hardware implementation for the 80386 architecture.

4.1 Description of Hardware

The hardware consists of two buffers, four comparators, two multiplexers and two perceptron networks. The Characteristic Buffer is a 4x20-byte storage buffer as shown in Figure 3. It stores all parameters associated with a loop. PI-Buffer is a duplicate of instruction fetch buffer, and keeps track of the instruction previous to current instruction in the instruction fetch buffer. Neural network N1 is a perceptron network that is trained to fire only when a backward jump instruction is encountered. Comparators C1 to C4 are required to compare operands of an instruction with determs. This is needed during identification of *determ changers*. Multiplexers M1 and M2 forward numerical factors from determ changers to neural network N2. Neural network N2 is a perceptron network that consists of two summation circuits N2a and N2b, followed by a threshold circuit T.

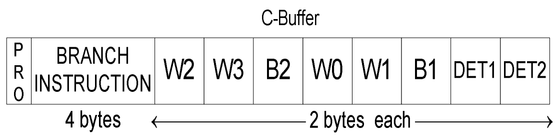

Fig.3. Memory required by the algorithm

4.2 Hardware operation

The contents of IF buffer are input to the first neural network (N1) that fires when it encounters the opcode of a branch instruction. This activates the Characteristic Buffer *(C-buffer)*, where a search is made

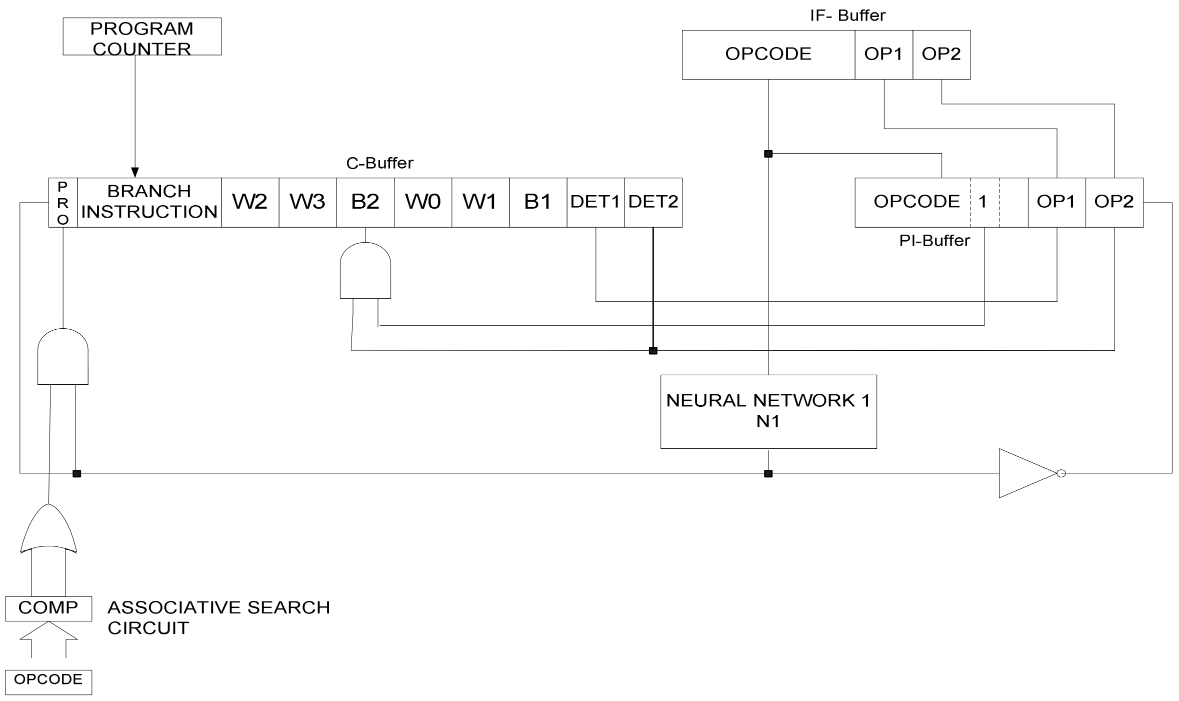

Fig.4. Branch processing circuit

to determine if an entry for the instruction is already present. When not found, the output of associative circuitry would be high, which activates a new entry in the C-Buffer. Branch program counter value, operands of previous instruction (identified as the *comparison statement)* and terminal value, if one determ is immediate, are filled in the entry. The *Pro*-bit or the Processing Bit is made high, which indicates that this entry is under processing by neural network. In the subsequent iteration, Pro-bit acts as the Enable to comparators C1-C4 that compare operands of incoming instructions in IF buffer with determs determined. A high on output lines of any one of the comparators indicates a *determ changer*, and its opcode is analyzed by the multiplexer M1 (or M2) to output the factor by which the determ changes. For e.g. if R1 is a determ, and the statement SUB R1, 5 is encountered, the corresponding multiplexer outputs the factor -5. A program might have one or more linear factors that change the value of determ. All these factors are latched to inputs to the summation circuit of respective perceptron. When the comparison statement encountered during processing stage is evaluated by the execution unit, it fetches values of determs, and hence they are available on data bus. By snooping these values in processing stage, the values B_1 and B_2 are set. Once the first iteration is totally traversed, output of N2b acts as threshold value in T.

This completes the branch processing. Whenever the branch is encountered again, buffers B1 and B2 are clocked high, and values of determs are updated and checked by perceptron structure. Output of perceptron is the branch prediction. “One” indicates that loop is not terminated yet, while “Zero” indicates that loop will terminate in this iteration and hence the branch should not be taken.

For nested loops, during the processing of an outer loop, if a branch instruction of an inner loop is encountered, the entry for that branch instruction is already found in the C-buffer and need not be processed again. The perceptron network snoops the new values of determs and simply goes into action for every iteration of that inner loop to predict the inner loop termination point.

5. Results

The algorithm was simulated and tested with assembly language program inputs. The prediction accuracies are as shown in Table 1.

Programs	No. of iterative loops detected/total no. of loops	No. of detected loops analyzed correctly	% of terminal points predicted correctly
fib.asm	1 / 1	1	100
search.asm	6 / 11	5	83
Qsort.asm	10/10	8	80

Table.1. Simulation Results

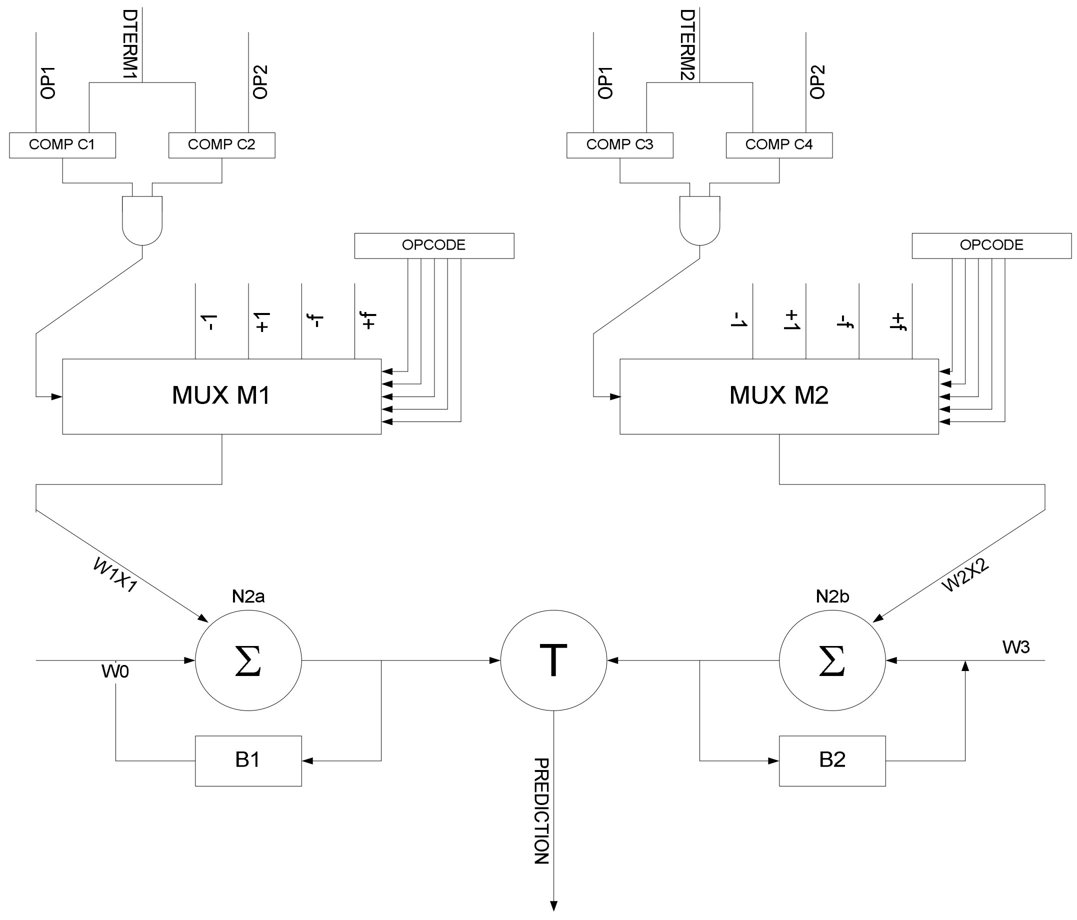

Fig.5. Branch predicting circuit

Each detected iterative loop corresponds to one misprediction by the current branch prediction algorithm, at its terminal point of these, the number of loops analyzed correctly correspond to the number of mispredictions saved by the proposed algorithm. Thus, the percentage of the improvement in Table 1 will be added to the accuracy proportions of existing branch predictors.

6. Conclusions and Future work

With pipelines becoming deeper and deeper, a stall penalty becomes more expensive. The pipeline stalls occur due to the time delay between the stages where branch information is required (i.e. IF stage) and where it is available (i.e. Execute Stage). We proposed a new branch prediction algorithm. Study of simple loop structures reveals certain parameters that give information about their characteristics. By gleaning these characteristics at the fetch stage, we feed them to the neural network which takes a decision about the direction the loop terminating branch condition will take. The algorithm has been coded in software using ‘Python 2.3’. The hardware for the algorithm has been designed. The results for three standard assembly level programs have been tabulated.

A basic prototype to correctly predict the loop terminating condition has been developed. The algorithm and the hardware developed work only for arithmetic operations. These can be extended to accommodate logical instructions in the future. Also, this algorithm takes care of only ‘for’ and ‘while’ loops. Further investigation can be carried out to correctly predict all other types of loops and branches. Branch misprediction rates would decrease if this hardware can be integrated with the currently available branch predicting hardware. Branch predictors using neural networks would be ideal to be interfaced with the above hardware scheme. An interfacing logic is required to interface our hardware to the existing hardware components. This interface should be designed in such a way that it should be able to distinguish between the predictions by the main predictor and this hardware. Further research is needed to design such an interface.

7. References

1. J. Smith, A Study of Branch Prediction Strategies, *Proc. 8th Annual Intl. Symp. on Computer Architecture*, May, 1981, pp. 135-148.
2. S. McFarling, Combining Branch Predictors, TR, Digital Western Research Laboratory, Jun. 1993.
3. T. Yeh and Y. Patt, Two-Level Adaptive Training Branch Prediction, *Proc. 24th Annual ACM/IEEE Intl. Symp. And Workshop on Microarchitecture*, Nov. 1991, pp. 51-61.
4. J. Lee and A. Smith, Branch Prediction Strategies and Branch Target Buffer Design, *Computer* 17:1 Jan. 1984, pp. 6-32.
5. S. McFarling and J. Hennessy , Reducing the Cost of Branches , *Proc. of 13th Annual Intl. Symp. OnComputer Architecture,* Jun. 1986*,* pp. 396-403.
6. J. Fisher and S. Freudenberger, Predicting Conditional Branch Directions from Previous Runs of a Program, *Proc. 5th Annual Intl. Conf. on Architectural Support for Prog. Lang. and Operating Systems,* Oct. 1992, pp. 85-95.
7. S. Pan, K. So, and J. Rahmeh, Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation, *Proc. 5th Annual Intl. Conf.on Architectural Support for Prog. Lang. and Operating Systems*, Oct. 1992, pp. 76-84.
8. T. Yeh and Y. Patt, A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History, *Proc. 20th Annual Intl. Symp. On Computer Architecture*, May 1993, pp. 257-266.
9. T. Ball and J. Larus, Branch Prediction for Free, Proceedings of the *ACM SIGPLAN '93,Conference on Programming Language Design and Implementation, 1*993, pp. 300-313.
10. N. Jouppi and D. Wall, Available Instruction- level Parallelism for superscalar and Superpipelined Machines, *Proceedings of ASPLOS III,* April 1989, pp. 272-282.
11. [11].D. Wall, Limits of Instruction-level Parallelism, *Proceedings of ASPLOS IV*, April 1991, pp. 176-188.
12. C. Young and M. Smith, A Comparative Analysis of Schemes for Correlated Branch Prediction, Proc. 22nd Annual Intl. Symp. On Computer Architecture, June1995 pp. 276-286.
13. C. Young and M. Smith, Improving the Accuracy of Static Branch Prediction Using Branch Correlation, Proc. 6th Intl. Conf. on Architectural Support for Prog. Lang. and Operating Systems, October 1994, pp. 232-241.
14. N. Gloy, M. Smith, and C. Young, Performance Issues in Correlated Branch Prediction Schemes, Proc. 28th Annual IEEE/ACM Intl. Symp. On Microarchitecture, Nov. 1995, pp. 3-14.
15. G. Steven, R. Anguera, C. Egan, F. Steven and Lucian Vintan, Dynamic Branch Prediction using Neural Networks, Proc. of Euro-DSD, September 2001, pp. 178-185.
16. Daniel A. Jimenez and Calvin Lin, Dynamic branch prediction with perceptrons, Proc. of Seventh International symposium in High Performance Computer Architecture, January, 2001, pp. 197-206.
17. Marcos R. de Alba and David R. Kaeli, Run time Predictability of Loops, IEEE 4th Annual Workshop on Workload Characterization, December, 2001, pp. 91-98.