Neural network based branch prediction unit in modern microprocessors

Dr. Faiz A. AL – Alawy

Iraqi Commission for Computers and Informatics

Abstract

For the modern microprocessors, as pipelines get deeper, or issuing rate gets higher, the penalty imposed by branching instructions gets larger. To reduce this penalty, branch prediction is used. Branch prediction unit is an important part of modern processor architectures. Its responsibility is to predict whether branches will be taken or not taken before they are actually executed.

The application of ANNs have been considered in this work as a good alternative for solving the problem of branch prediction. Single and multilayer preceptron neural nets have been used to design a new branch predictor. The designed neural nets have been tested for different applications.

A comparative analysis and study have been carried out with the other known prediction techniques. The achieved results show very high prediction accuracy.

The prediction accuracy rates are calculated for different types of neural predictors and conventional predictors. It has been concluded that the neural predictors are better than conventional predictors, but in the other side, when using adaptive techniques, the neural predictors are comparable to conventional two-level predictors with the same size of input. Regarding the same hardware budget, neural predictors are the best, but they might take more time for computing branch prediction than conventional predictors.

1. Introduction

Branch instructions permit a program to control what instructions are executed. If then, and looping instructions represent the main examples of conditional instructions. They test some conditions, and depending on the outcome, execution proceeds down one of two possible paths. Branch instructions have exactly two possible outcomes: not-taken, the sequential case in which the program continues executing the instructions that immediately follow the branch, and taken, the non-sequential case in which execution jumps to a target specified in the branch instruction. The target can lie anywhere within the program.

Other control flow instructions can transfer execution to some other program location but are not conditional. These jump instructions either jump to the target specified in the instruction (direct jumps), or jump to a target that has been computed and whose address is found in a register (indirect jumps). A procedure call is an example of the former, and a procedure return is an example of the latter. As with branches, some time is required to determine jump targets. Direct jumps can be resolved early with proper hardware in the fetch stage to extract the jump target from the instruction, or the targets that can be predicted. Indirect jumps generally cannot be resolved early, and instead must proceed through the pipeline in order to read their target from the register file, just like any other instruction. Fortunately, their targets can also be predicted.

The ratio of branches accounts for about 20% from instruction in general programs. This means on average, each fifth instruction is a branch, and the majority of branches (approximately 80%) are conditional [1].

2. Branch Prediction Techniques

Branch prediction techniques are classified as static or dynamic [1, 2].

Static prediction schemes can be regarded the simpler class. The most straightforward type is to predict the branch to be always taken by observing that majority of branches is taken. Static schemes can also be based on branch op-codes. Another simple method is using the direction if the branches to make a prediction. If the branch is backward, i.e., the target address (decrementing), it is predicted to be taken. Otherwise, if the branch is forward, the prediction is not to be taken. This strategy tries to take advantage of loops in the program. It works well for programs with many looping structures. However, it does not work well in the case where there are many irregular branches. Profiling is another static strategy which uses previous runs of a program to collect information on the tendencies of a given branch to be taken or not taken and preset a static prediction bit in the op-code of the given branch. Later runs of the program can use this information to make predictions. This strategy suffers from the fact that runs of a program with different input data sets usually result in different branch behaviors. While dynamic prediction may change to reflects the time-varying activity of the program.

The second class is the dynamic branch prediction, which make use of the information gathered at run-time to predict branch direction. There are several dynamic branch predictors in use or being researched nowadays. Those include One-Level branch predictors, Two-Level branch predictors and Hybrid predictors.

(2-1) Branch Target Buffer (BTB)

It is a cache indexed by instruction address that stores the target address for the most recently taken branches. When an instruction is fetched [3], the same address is offered to BTB, if there is a match in BTB, the next instruction is fetched using the target address specified in the BTB if branch is predicted as taken.

(2-2) One-Level Branch Prediction

The most basic mechanism is a simple table of binary values, one per branch. This value is updated to provide the last outcome witnessed for each branch, and so each time a branch changes direction from taken to not-taken or vice-versa a misprediction results. Making the table entries not-taken outcome decrements the counter (until it hits 00), and each taken outcome increments it (until it hits 11). Values of 00 and 01 produce a not-taken prediction, and values of 10 and 11 produce a taken prediction.

(2-3) Two-Level Branch Prediction

More bits can be added to the two-bit counters, but too many bits make it difficult for the predictor to learn legitimate changes in direction. A better refinement is to explicitly track branch history patterns, and each branch makes different predictions depending on the

recent history. These predictors are called two-level predictors. A local history predictor keeps a table of shift registers, one entry per branch. To make a prediction, the predictors looks up the branch's history and then uses the history to index the now-familiar table of two-bit counters. These counters now track the taken/not-taken behavior of branch history patterns, and not the overall behavior of individual branches. This permits common patterns, like alternating branches (TNTN…), to be correctly predicted, and can also learn irregular patterns that correspond to some program or input data behavior (e.g. TTTNTTTN…) [3].

3- Conventional Predictors

Some particular configurations will be described her in order to be used as a base for the proposed predictor.

(3-1) Global Adaptive (GA) Predictor

As shown in Figure 1, GAg uses a single global branch history register (GBHR) that records the outcome of the last K branches encountered, and a single global pattern history table (PHT) containing an array of prediction counters. To generate a prediction, the k bit pattern in the first level GBHR is used to index the array of two-bit saturating prediction counters in the second level PHT. Each branch prediction seeks to exploit correlation between the next branch outcome and the outcome of the k most recently executed branches. The GBHR and the prediction counter in the PHT are updated as soon as the branch is resolved. Unfortunately, since all the branches in GA predictor share a common set of prediction counters in the PHT, the outcome of one branch may interfere with the prediction of all other branches. To solve this problem, a two direction PHT is used and this type is called Gas. The PHT has its rows indexed by the GBHR and its columns indexed by the branch address. The GBHR is n-bits wide register used to address the rows of the PHT.

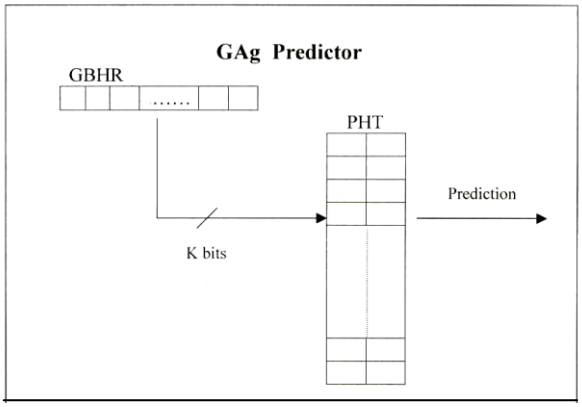

Figure 1: Diagram of the GAg predictor.

(3-2) Gshare Predictor

As shown in figure 2, Gshare scheme attempts to reduce interference by randomizing the index to second level table through xor-ing the GBHR with branch addresses, the gshare scheme can produce new distinct indexing values for counters, each associated with a static branch. This xor-ing can reduce interference between branches while retaining the advantages of using long GBHR to exploit branch correlation. However, this scheme offers limited benefits, because randomization can only "blindly" separate alised branches. Consequently, this process may reduce destructive interference simple by chance.

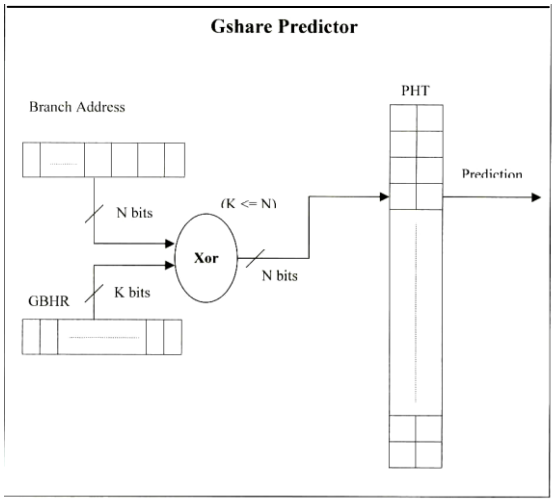

Figure 2: Diagram of the Gshare predictor.

(3-3) Per-Address Adaptive (PA) Predictor

The architecture of PAg consists of two tables. The first-level table, called BHT, has multiple shift-registers. Each of these registers is used to record past branch outcomes for a single static branch. The branch outcome patterns recorded in BHT are then used to index in second level, which is a single global PHT. Or a two direction table which is called branch history table (BHT) as shown in Fig.3 which is known as PAs.

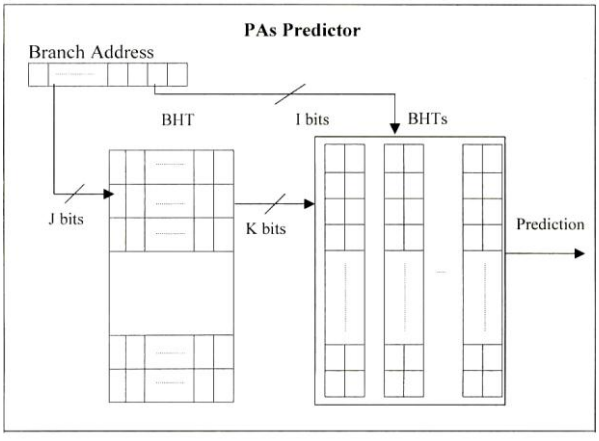

Figure 3: Diagram of the PAs predictor.

(3-4) Hybrid Predictors

To further improve prediction accuracy, hybrid branch predictors have recently been proposed [4, 5, 6, 7]. Within most programs, some branches are best predicted using global history, while others are best predicted using local history. A hybrid branch predictor is composed of two or more single-scheme predictors and a mechanism to select among these predictors.

4-system design

A performance measuring tool, a trace program (TP) which is written in assembly language has been designed to operate the processor in a single step mode through providing the ability to execute one instruction at a time. Also, this program is able to test the contents of registers or memory both before and after the execution of each instruction.

Five test programs have been organized to be used as case study to measure the performance for different branch prediction strategies.

(4-1) Neural Predictor design.

Single layer and Multi layers perceptron neural networks have been used to design the neural predictor.

The neural net is used to replace the PHT (2-bit counter) for the two-level adaptive predictor. The input to this net can be either (-1) for not taken or (1) for taken.

Fig. 4.a shows the block diagram of the suggested SLP predictor.

When the actual outcome of branch becomes known, the training algorithm uses this outcome with the output to update the weights of

the selected perceptron. These updated weights are written back to the perceptron table. Weights updating operation are carried out, only, when the prediction result is not true.

Fig. 4.b shows the MLP predictor. The least significant bits of the branch address with the history register have been used as inputs to the MLP net.

Bipolar sigmoidal function has been used as activation function for the neurons.

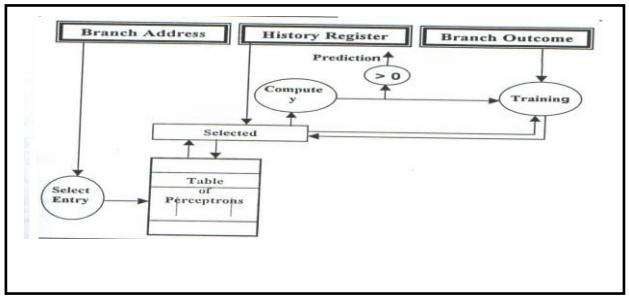

Fig 4.a: The organization of SLP NNs Predictor

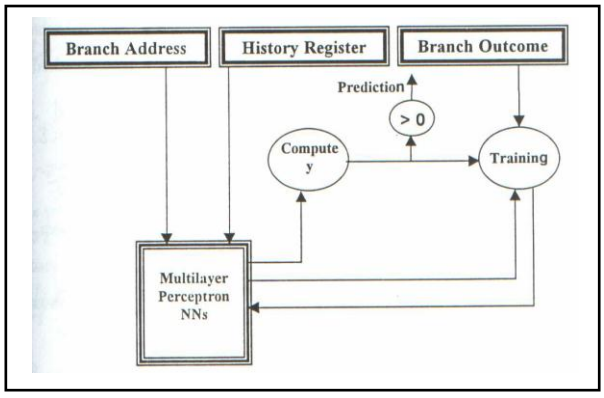

Fig. 4.b: The organization of MLP NNs Predictor

The number of the hidden neurons used is half the number of input neurons.

(4-2) Other implemented alternatives

Different types of dynamic input information for the neural predictor have been tried to implement a new combination of neural predictors.

These are:

GAs, PAs and GPA with single layer perception neural networks and also with multi layer perception.

(4-3) Estimated hardware budget

The hardware budget of each implemented predictor can be estimated depending on the number of bits for first level and every weight used in the second level. For SLP, each weight value is implemented by a single byte, and for MLP it is implemented by 4 bytes.

Table 1; summarize the hardware cost for all discussed prediction techniques.

(4-4) Performance measure

A comparative study between all predictor types and with the new suggested types have been carried out using the same five testing programs.

Prediction accuracy represents one of the most important measures for the predictors. It can be computed as follows: -

Figure 5 shows the average of the prediction accuracy rates achieved for different configurations of neural network predictors and conventional predictors. When using the global information, neural predictors performs better than the conventional one, but for per-address predictors all the predictors have, nearly, the same performances.

	i	j	k	First level	Cost	Second level	Cost	AV.occur act
G(A)g-4K bits			11	11	11	2048x2	4096	93.079
G(A)g-8K bits			12	12	12	4096x2	8192	93.531
G(A)s-4K bits	2		9	9	9	512x4x2	4096	93.332
G(A)s-8K bits	2		10	10	10	1024x4x2	8192	93.767
P(A)g-4K bits		2	11	4x11	44	2048x2	4096	94.110
P(A)g- 8K bits		2	12	4x12	48	4096x2	8192	93.813
P(A)s-4K bits	2	2	9	4x9	36	512x4x2	4096	93.785
P(A)s-8K bits	2	2	10	4x10	40	1024x4x2	8192	94.023
G(A)slp-4K bits	6		7	7	7	64x8x8	4096	93.621
G(A)slp-8K bits	6		15	15	15	64x16x8	8192	94.538
P(A)slp-4K bits	6	2	7	4x7	28	64x8x8	4096	94.107
P(A)slp-8K bits	6	2	15	4x15	60	64x16x8	8192	94.300
GP(A)slp- 4K bits	6	2	7	3+4x4	19	64x8x8	4096	94.300
GP(A)slp-8K bits	6	2	15	5+4x10	45	64x16x8	8192	93.813
G(A)mlp-4K bits	8		7	7	7	(16x7+8) x32	3840	94.479
G(A)mlp-8K bits	8		13	13	13	(22x10+1 1)x32	7392	92.894
P(A)mlp-4K bits	8	2	7	4x7	28	(16x7+8) x32	3840	93.612
P(A)mlp-8K bits	8	2	13	4x13	52	(22x10+1 1)x32	7392	93.749
GP(A)mlp-4K bits	8	2	7	3+4x4	19	(16x7+8) x32	3840	93.732
GP(A)mlp-8K bits	8	2	13	3+4x10	43	(22x10+1 1)x32	7392	93.188

Table 1: The cost of hardware for neural and conventional predictors.

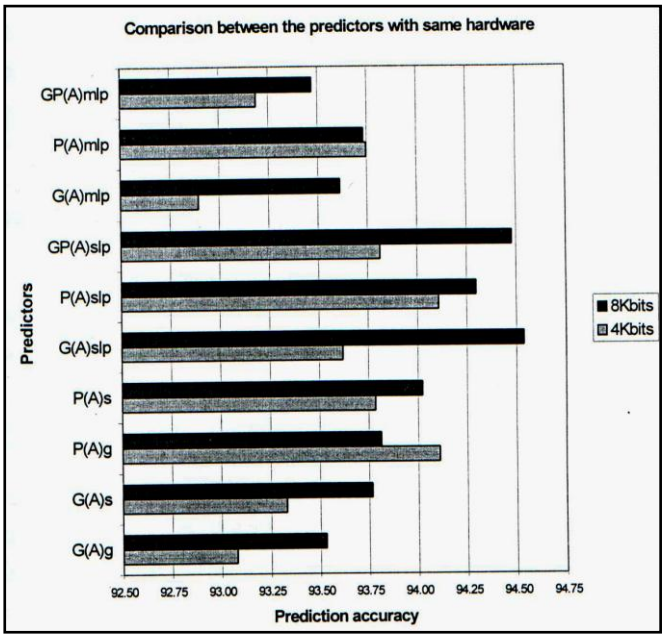

Fig. 5 Comparison of Neural and conventional predictors with the same cost of hardware.

(4-6) Hybrid predictors

In this work two types of predictors, Gshare and neural, for the same history length have been combined to get one predictor, This hybrid predictor is tested as a dynamic predictor for PHT with 1024 entry of two-bit saturating counters.

Fig. 6 shows how this combination of Gshare and MLP can improve the performance of the MLP alone in a percentage higher than the Gshere when it is combined with the SLP.

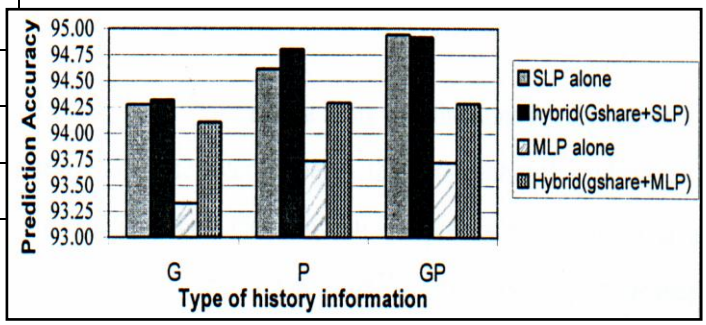

Fig.6 Performances, when using hybrid predictors

(5) Conclusions

It can be concluded that:

1) For non-adaptive mechanisms with the same history register length the performance of neural net predictor is better than the performance of conventional predictor. The neural net predictor is better than conventional predictor with global information, but the conventional predictor is better than neural net predictor with local information.

2) Neural predictors can handle more history of branches than the conventional.

3) Predictors with SLP can out perform others with MLP. That means that there is no

need to more compicated design in order to get higher accuracy.

4) The operation of MLP predictor is more time consuming than the SLP, it takes more time to predict, and also it takes more time for weights adjustment operation.

References

1) D. Sima, T. Fountain and P. Kacsuk. "Advances Computer Architecture", A design space approach. Addison Wesley, 1998.

2) A. Malishevky, D. Beck, A. Schmid and E. Landry, "Dynamic Branch Prediction". Htt:/www.ece.orst.edu/~benl/projects/branch-pred/

3) T. Yeh and Y. Patt, "A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution". In Proceedings of the 25th International Symposium Microsrchitecture, Portland, Oregon, November, 1992.

4) Scoot Mcfarling. "Combining Branch Predictors". Technical Report TN-36, Digital Western Researcg Laboratory, June 1993.

5) P.-Y. Chang, E. Hao, T.-Y. Yeh and Y.N Patt. "Branch Classification: a New Mechanism for Improving Branch Predictor Performance. In proceedings of the 27th Annual ACM/IEEE International Symposium on Microarchitecture, PP. 22-31, 1994.

6) P.-Y. Chang, E. Hao, and Y.N. Patt. "Alternative Implementations of Hybrid Branch Predictors". In proceddings of the 28th Annual International Symposium on Computer Architecture, June 1995.

7) M. Evers, P.-Y. Chang and Y.N. Patt. "Using hybrid branch predictors to improve branch prediction accuracy in the presence if context switches". In proceedings of the 23rd International Conference on Computer Architecture, May 1996.