Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Journal of Systems Architecture 49 (2003) 557–570

JOURNAL OF SYSTEMS ARCHITECTURE

www.elsevier.com/locate/sysarc

Two-level branch prediction using neural networks

Colin Egan [a,*], Gordon Steven [a], Patrick Quick [a], Rubén Anguera [a], Fleur Steven [a], Lucian Vintan [b]

[a] *University of Hertfordshire, College Lane, Hatfield AL10 9AB, UK*
[b] *University "Lucian Blaga" of Sibiu, Sibiu-2400, Romania*

Abstract

Dynamic branch prediction in high-performance processors is a specific instance of a general time series prediction problem that occurs in many areas of science. Most branch prediction research focuses on two-level adaptive branch prediction techniques, a very specific solution to the branch prediction problem. An alternative approach is to look to other application areas and fields for novel solutions to the problem. In this paper, we examine the application of neural networks to dynamic branch prediction. We retain the first level history register of conventional two-level predictors and replace the second level PHT with a neural network. Two neural networks are considered: a learning vector quantisation network and a backpropagation network. We demonstrate that a neural predictor can achieve misprediction rates comparable to conventional two-level adaptive predictors and suggest that neural predictors merit further investigation.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Neural branch prediction; Two-level adaptive branch prediction; Backpropagation network; Learning vector quantisation network

1. Introduction

Branch instructions are a major bottleneck in the exploitation of instruction level parallelism (ILP). The proportion of conditional branch instructions in a program is relatively high. For example, in general-purpose code conditional branches occur approximately every 5–8 instructions [13]. In a simple processor, instructions from the sequential path are typically pre-fetched from the instruction cache in an attempt to ensure that its pipeline is fully utilised. A misfetch penalty occurs when a branch is taken and incorrect instructions have been fetched into the pipeline. The pipeline is then flushed and reloaded with instructions from the branch target. As more pipeline stages are introduced, the misfetch penalty increasingly degrades processor performance. Since there is such a high frequency of branches in general-purpose code, it is essential to reduce the performance degradation by using accurate branch prediction to pre-fetch the correct instructions into the pipeline. A misprediction penalty then only occurs if the branch prediction mechanism incorrectly predicts the branch destination.

* Corresponding author. Tel.: +44-1707-284373; fax: +44-1707-284303.
E-mail address: c.egan@herts.ac.uk (C. Egan).

The branch prediction problem, therefore, consists of two sub-problems: firstly generating the correct prediction and secondly in the case of a taken branch predicting the correct branch target.

With ever increasing issue rates in multiple-instruction issue (MII) processors and deeper pipelines the impact of a branch misprediction will severely limit any gains in processor performance. Very high prediction accuracy is required because an increasing number of instructions are lost before a branch misprediction can be corrected. As a result even a misprediction rate of a few percent involves a substantial performance loss. Even a 3% misprediction rate, achieved by current state-of-the art two-level adaptive branch predictors, can have a severe limiting impact on MII processor performance [22]. Some researchers [2,22] envisage that a next generation branch predictor could consume 1 Mbyte of the hardware budget. This suggests that current and near future branch predictors, though highly effective in prediction accuracy, are not cost effective. Future branch predictors must sustain prediction accuracy as close to 100% as possible and yet be more cost effective.

If branch prediction is to improve performance, branches must be detected within the dynamic instruction stream, and both the direction taken by each branch and the branch target address must be correctly predicted. Furthermore, all of the above must be completed in time to fetch instructions from the branch target address without interrupting the flow of new instructions to the processor pipeline. A classic branch target cache (BTC) [13] achieves these objectives by holding the following information for previously executed branches: the address of the branch instruction, the branch target address and information on the previous outcomes of the branch. Branches are then predicted by using the PC address to access the BTC in parallel with the instruction fetch process. As a result each branch is predicted while the branch instruction itself is being fetched from the instruction cache. Whenever a branch is detected and predicted as taken, the appropriate branch target is available at the end of the instruction fetch cycle, and instructions can be fetched from the branch target in the next cycle. Straightforward prediction mechanisms based on the previous history of each branch give a prediction accuracy of around 80–95% [13]. This success rate proved adequate for scalar processors, but is generally regarded as inadequate for superscalar designs.

The requirement for higher branch prediction accuracy in superscalar systems and the availability of additional silicon area led to a dramatic breakthrough in the early 1990s with the development of two-level adaptive branch prediction [21,33]. More recently two-level branch predictors have been implemented in several commercial microprocessors [15,16]. However, although high prediction rates are achieved with two-level adaptive predictors, this success is obtained by providing very large arrays of prediction counters or pattern history tables (PHTs). Since the size of the PHT increases exponentially as a function of history register length, the cost of the PHT can become excessive, and it is difficult to exploit a large amount of branch history effectively. Two-level adaptive branch predictors have two other disadvantages. Firstly, in most practical implementations each prediction counter is shared between several branches. There is therefore interference or aliasing between branch predictions. Secondly, large arrays of prediction counters require extensive initial training before they can predict accurately. Furthermore, the amount of training required increases as additional branch history is exploited, further limiting the amount of branch history that can be exploited.

Finally, some branches remain stubbornly hard to predict [19,28]. There are two cases. The outcome of some data dependent branches is effectively random and these branches will never be accurately predicted. However, it should be possible to predict certain branches that are currently hard to predict more accurately by identifying new correlation mechanisms and adding them to the prediction process. We suggest that neural predictors may prove to be a useful vehicle for investigating potential new correlation mechanisms.

We emphasise that most branch prediction research is based on two-level adaptive branch predictors, which are themselves based on two closely related correlation mechanisms. Yet, branch prediction is a specific example of a general time series prediction problem that occurs in many diverse fields of science. It is therefore surprising that there has not been more cross-fertilisation of ideas between different application areas. A notable exception is a paper by Mudge's group [7] that demonstrates that all two-level adaptive predictors implement special cases of the prediction by partial matching (PPM) algorithm that is widely used in data compression, speech recognition and handwriting recognition problems. Mudge uses the PPM algorithm to compute a theoretical upper bound on the accuracy of branch prediction. In a later paper, Steven et al. [27] demonstrate that a two-level predictor can be extended to implement a simplified PPM algorithm with a resultant reduction in the misprediction rate. Time series prediction is also an important research topic in neural networks (NNs). It therefore appears natural to look to NNs for a further cross-fertilisation of ideas.

In this paper we explore how NNs can be used to dynamically predict branch outcomes by forecasting future values of data series. One of our main research objectives is to use NNs to identify new correlations that can be exploited by branch predictors. We also wish to determine whether more accurate branch prediction is possible and to gain a greater understanding of the underlying prediction mechanisms. In this paper, we apply NNs to dynamic branch prediction to demonstrate that NNs can achieve the same prediction accuracy as a conventional two-level adaptive predictor. We therefore restrict our neural network inputs to using the same dynamic history register (HR) information as a conventional two-level predictor. Finally, we hope to design and evaluate hardware implementations of simplified neural branch predictors, although in this initial feasibility study, we have ignored the costs of implementing the NNs, assuming that the predictions would be produced in time to be useful. Alternatively, our research may lead to the design of more effective two-level branch predictors.

We explore the suitability of two NNs, a learning vector quantisation network (LVQ) and a backpropagation network, for branch prediction. Through trace-driven simulation, we demonstrate that neural predictors can achieve success rates that are comparable to conventional two-level adaptive predictors.

2. Related work

2.1. Two-level adaptive branch prediction

Most recent research on branch prediction has focused on two-level adaptive prediction [6,11,17,18,21,23,24,33–36]. In a two-level predictor, the first level consists of a HR that records the outcome of the last k branches encountered. The HR may be a single global register (HRg) that records the outcome of last k branches executed in the dynamic instruction stream or one of multiple local HRs (HRl) that records the last k outcomes of each branch. The second level of the predictor, known as the PHT, records the behaviour of a branch during previous occurrences of the first level predictor. The PHT typically consists of an array of two bit saturating counters that is indexed by the HR to obtain the prediction. With a k-bit HR, 2^k entries are therefore required if a global PHT is provided, or many times this number if separate HRs and therefore PHTs are provided for each branch.

Two distinct prediction techniques have in fact been developed, global and local (per-address). If a global HR is used, the predictor exploits correlation between the outcome of a branch and the outcome of neighbouring branches that are executed immediately prior to the branch. If a local HR is used, the predictor exploits correlation between the outcome of a branch and previous outcomes of the same branch.

Two-level branch predictors are usually classified using a system proposed by Yeh and Patt [34]. The six most common configurations are GAg, GAp, GAs, PAg, PAp and PAs. The uppercase first letter specifies the first-level mechanism, "G" global or "P" local (per-address). The lowercase last letter specifies the second level, which can be "g" global, "p" local (per-address) or "s" a set of branches mapping to the same PHT prediction array. The "A" in the middle emphasises the adaptive or dynamic nature of the predictor. Therefore, GAg, GAp and GAs rely on global branch history and PAg, PAp and PAs rely on local branch history. Additionally a separate BTC is still required to provide branch target addresses. In the case of a local predictor the local HRs can be integrated into the BTC by adding a HR field to each BTC entry.

Since most recent research on branch prediction has concentrated on two-level adaptive techniques, it is useful to explore some of the drawbacks of two-level predictors. The main disadvantages can be found in the following areas:

- High cost of PHT
- Branch interference
- Slow initialisation

The high cost of two-level predictors is a direct result of the size of the second level PHT, which increases exponentially in size as a function of HR length. The high implementation cost of conventional two-level predictors has had a subtle, but important, impact on branch prediction research; it has discouraged any developments that increase the size of the PHT. Perhaps the most obvious example is that researchers are deterred from attempting to extract additional prediction accuracy from very long HRs. Similarly, researchers are discouraged from describing the program path leading to each branch more accurately by recording fuller path information [20], and from combining global and local history information in a single predictor. The use of additional prediction information, such as the branch direction information used in the simple backwards taken, forward not taken (BTFNT) heuristic, is discouraged.

The high cost of a conventional PHT suggests that alternative configurations should be considered. One possibility is to replace the PHT with a prediction cache [8–10,27]. Although a tag field must be added to each PHT entry, the very large size of conventional PHTs suggests that the total number of entries and therefore the total cost of the PHT could be significantly reduced. The danger with a prediction cache is that cache misses will increase the misprediction rate. The impact of prediction cache misses can, however, be minimised, at low cost, by restoring a two-bit prediction counter to each entry of the conventional BTC, which is still required in all two-level predictors to furnish the target address for each branch. These BTC counters can then be used to provide a default prediction whenever there is a miss in the prediction cache [8]. Alternatively, an entirely different approach, such as the neural branch predictors introduced in this paper, can be investigated.

In a global predictor, interference or aliasing in the second level PHT occurs whenever two branches generate the same first-level HR pattern and therefore access the same second-level PHT prediction counter [23]. This causes the predictions for two or more branches to affect each other. There have been numerous studies that attempt to reduce the impact of interference [5,10,17,18,24]. However, reducing the amount of interference has generally also increased the initialisation mispredictions and the cost of the predictor.

The third problem is PHT initialisation. In the worst case, the 2^k prediction counters associated with each branch, where k is the length of the HR, must be trained before the predictor is fully effective. Even allowing for the fact that a PHT is effectively a sparse matrix with many unused entries, this situation contrasts sharply with a classic BTC that is fully initialised after one execution of each branch. The impact of PHT training can be reduced by combining a two-level predictor with a classic BTC in a hybrid predictor [6,26]. The BTC will then be used in preference to the two-level predictor while the latter is being initialised.

2.2. Neural branch prediction

Calder introduced ideas from artificial intelligence to branch prediction by using a neural branch predictor to derive static branch predictions [3]. He termed this new technique 'Evidence-based Static Prediction' (ESP). However, Calder was concerned entirely with static or compile-time branch prediction. His predictions were therefore based on information about a program's structure that was readily determined by a compiler. For example, a branch successor path that leads out of a loop or function is less likely to be followed than a path that remains within the loop or function. The idea of Calder's neural branch predictor was to map static features associated with each branch to the probability that the branch will be taken. This approach provides several advantages over the traditional static branch prediction program-based heuristics [1]. First, the technique can be applied to a large range of programming languages, compilers and architectures, since the neural net will perform a dynamic mapping of static features to the probability that the branch will be taken. Second, since the technique uses artificial intelligence algorithms, the most useful heuristics are automatically applied to derive the prediction in cases where more than one heuristic applies, making it unnecessary to order heuristics. Calder achieves a misprediction rate of 20%, remarkably low for static branch prediction. Since Calder's predictions were performed at compile time, he was unable to feed the dynamic branch histories used by two-level predictors into his NNs. As a result, perhaps the most useful contribution of his paper is to suggest a wide range of alternative inputs that might correlate with branch outcomes and which might therefore be usefully added to dynamic predictors.

Jiménez, independently and simultaneously to our investigation into the potential of applying the principles of NNs to dynamic branch prediction, has also investigated neural methods for dynamic branch prediction [14]. His study focuses on a perceptron predictor; we also focus on two perceptron predictors: a simple learning vector quantisation (LVQ) neural predictor and a backpropagation neural network predictor. The major difference between the two studies is that Jiménez only uses the history register as input values to his perceptron predictor, whereas we use the history register and the branch address as input values to our LVQ and backpropagation neural predictors. Furthermore, Jiménez claims to have developed the first perceptron predictor that successfully uses NNs. As far as we are aware, the first known perceptron predictor was developed by Vintan [29–32].

The perceptron and the Adeline [12] are two of the simplest models used by NNs for pattern recognition. A perceptron net consists of a single layer of neurons. Each of the inputs is connected to each neuron by one-way data connections, and each of these connections has an associated weight. The size of the weight is applied to its connection to determine a single output signal. Additionally, each neuron applies a threshold activation function to its input signals, known as the bias weight. It is important to use bipolar input signals (−1, 1) rather than binary input signals (0, 1) because a weight with an input signal of 0 would have no impact on the network. The output signal is the sum of bias weight with the summation of the product of each input signal and its associated weight.

$$\text{output\_signal} = w_0 + \sum_{i=1}^{n} w_i * \text{input\_signal}_i$$

In the case of branch prediction, a prediction is considered to be taken if the output signal is $\geqslant 0$ and a prediction is considered not-taken if the output signal is <0.

Both Jiménez and Vintan retained the first level history register of a two-level predictor to supply input signals to their perceptron predictors. However, Vintan's input signals consisted of the branch address as well as the history register. In both studies the neural network was dynamically trained after each branch prediction. Vintan applied the well-known training algorithm that is usually applied to the backpropagation learning algorithm [12]. The backpropagation learning algorithm has two steps. The first step is the forward propagation step that computes the weighted sums and activations for each input value. The second step is used to update the weight for each input value and is a backward pass through the neural network. In contrast, Jiménez used a simpler training algorithm. In his predictor an input value to the network was either −1 or +1. If the input value was the same as the previous branch outcome, where a not-taken branch is associated with −1 and a taken branch associated with $a + 1$, then the weight was incremented. Conversely, if the input value was the not same as the previous branch outcome then the weight was decremented.

Both researchers conclude that greater correlations are achieved by neural predictors than by two-level predictors and greater prediction accuracy can be achieved. Jiménez showed that his predictor achieved a misprediction rate of 1.71%, which equates to 36% fewer mispredictions than a McFarling style hybrid two-level predictor [18]. Vintan showed that his predictor achieved a misprediction rate of about 11%, which equates to 3% improvement in the misprediction rate for his neural predictor over a conventional two-level predictor. The difference in the prediction accuracy between Jiménez and Vintan study's may be explained by the fact that Jiménez focused his study on history register lengths as input values for his perceptron predictor and was able to achieve correlations for history register lengths up to 66 bits. In contrast, Vintan used a combination of the branch address and the history register length as input values into his multiplayer layer perceptron predictor, which restricted his maximum history register length of 10 bits. Furthermore, the difference in the training algorithms may be a critical factor in determining the behaviour of the NNs.

3. Branch prediction models

We first briefly consider the performance of a simple learning vector quantisation (LVQ) neural predictor. We then compare the performance of conventional two-level adaptive predictors with a neural predictor using a backpropagation network. In both cases the prediction process is based on two inputs: the branch PC's 10 least significant bits and the HRg, HRl or a combination of HRg and HRl of the k previous branches.

We assume that all predictions are being made during the instruction fetch (IF) stage of the processor pipeline. All our predictors therefore operate in parallel with a classic BTC that detects branches and furnishes the branch target address. Since an instruction is not decoded until the ID stage of the pipeline, then only hits in the BTC are known to be branch instructions. The actual prediction is generated by either a neural network or a two-level predictor. Nevertheless, a miss in a BTC always results in a default prediction of not-taken, irrespective of the prediction delivered by the predictor, since the instruction is not necessarily a branch. A 1K four-way set associative BTC is used throughout the paper.

3.1. An LVQ neural predictor

The first neural network we examined was an LVQ [12] model. Our objective was to determine whether respectable success rates could be delivered by a simple LVQ network that was dynamically trained after each branch prediction.

The LVQ predictor contains two "codebook" vectors: the first vector, Vt, is associated with the branch taken event and the second, Vnt, with the not-taken event. Vt is initialised to all ones and Vnt to all zeros. During the prediction process, the input parameters of the branch address (10 least significant bits) and the k bits of the HR are concatenated to form a single input vector, X. Modified Hamming distances are then computed between X and the two codebook vectors.

$$\mathrm{HD} = \sum_i (X_i - V_i)^2$$

The vector with the smallest Hamming distance is defined as the winning vector, Vw, and is used to predict the branch. A win for Vt therefore indicates "predict taken", while a win for Vnt indicates "predict not-taken". When the branch outcome is determined, the codebook vector Vw that was used to make the prediction is then adjusted as follows:

$$Vw(t+1) = Vw(t) + / - a(t)[X(t) - Vw(t)]$$

To reinforce correct predictions, the vector is incremented whenever a prediction was correct and decremented otherwise. The factor $a(t)$ represents the learning factor and is usually set to a small constant less than 0.1. In contrast, the losing vector is unchanged. The neural predictor will therefore be trained continuously as each branch is encountered. It will also be adaptive since the codebook vectors will always tend to reflect the outcome of the branches most recently encountered. In our LVQ predictors, the most recently encountered branches may be global, local or a combination of global and local depending on the HR information used as input signals.

3.2. Branch prediction using a backpropagation neural network predictor

Our second neural network is a backpropagation neural network [4]. The prediction information is fed into a backpropagation net (Fig. 1) which predicts the outcome of each branch. Later when the outcome of the branch is known, the error in the prediction is backpropagated through the neural network using the classic backpropagation algorithm.

In a backpropagation neural network there are two steps through the network [4]. The first step is a forward sweep from the input layer to the output layer, and then there is a backward step from the output layer to the input layer. The forward step propagates the input vector of the net into the first layer. Outputs from this layer produce a new vector, which is used as the input into the second layer. This procedure is repeated through to the final layer. The outputs of the final layer are the output signals of the network. In the case of a neural branch predictor there is only one output signal, which is used as the prediction. The backward step is similar to the forward step, except that error values are propagated back through the network. These error values are used to determine how the weights are changed. The back propagation algorithm is therefore used to generate the weights. Inputs and outputs of a backpropagation algorithm may not be restricted to discrete values which results in a wider range of ways to codify inputs and outputs.

Four different networks have been developed: two of them use global history information while the other two use local history information. The two versions of each arise because the inputs to the net are coded in two different ways: *binary* using 0 and 1 for not-taken and taken branches respectively, and *bipolar*, using −1 and 1. To exploit these different input encodings, two different activation functions are also required, a sigmoidal function for binary inputs and a bipolar sigmoidal function for bipolar inputs:

$$\text{Sigmoidal function}: 1/(1+e^{-\beta x})$$

$$\text{Bipolar sigmoidal function}: 2/(1+e^{-\beta x})-1$$

The factor β controls the degree of linearity in the two activation functions. In particular, as β approaches infinity, the functions become step functions, the form of the activation function used in multi-layer perceptron (MLP) networks.

When predicting a branch using binary inputs, a value greater than or equal to 0.5 on the output cell is considered to be a taken prediction, whereas any value lower than 0.5 is a not-taken prediction. In the bipolar case, positive values or 0 indicate a taken prediction and negative values not-taken. The network is not initially trained, so random values are chosen for the weights between $[-2/X, 2/X]$, where X corresponds to the number of inputs to the net. This selection of weights guarantees that both the weights and their average value will be close to zero and that the net is not biased initially towards either taken or not-taken.

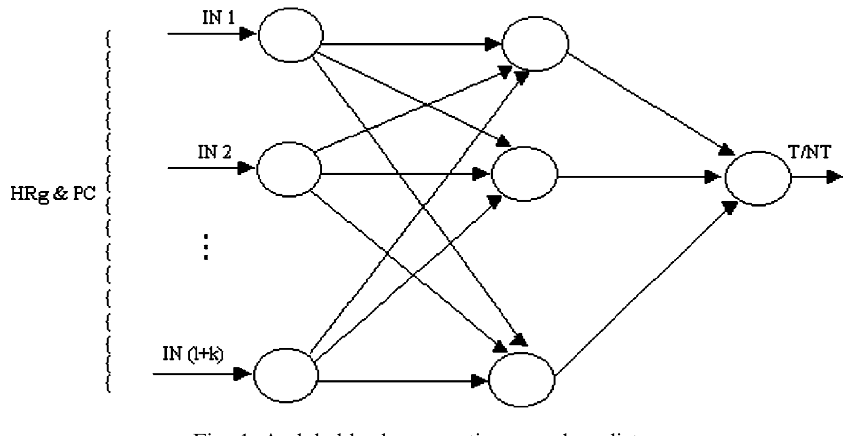

Fig. 1. A global backpropagation neural predictor.

4. Trace-driven simulation results

4.1. Simulation environment

Our simulations used the Stanford integer benchmark suite, a collection of eight C programs designed to be representative of non-numeric code, while at the same time being compact. The benchmarks are computationally intensive, with an average dynamic instruction count of 273,000. About 18% of the instructions are branches of which around 76% are taken. Some of the branches in these benchmarks are known to be particularly difficult to predict; see for example Mudge's detailed analysis [19] of the branches in *quicksort*. Branch prediction researchers generally use the SPEC benchmark suite but we consider that the SPEC benchmark suite may not necessarily be the most suitable benchmarks for branch prediction. The programs in the Stanford integer benchmark suite are shorter than those in the SPEC benchmark suite. This means that each branch in the Stanford integer benchmark suite is executed fewer times than each branch in the SPEC benchmark suite. Branches in the Stanford benchmark suite are therefore more difficult to predict because the initial training problems are more acute than those in the SPEC benchmark suite and thus provide a better test of prediction accuracy.

The Stanford benchmarks were compiled using a C compiler developed at the University of Hertfordshire for the Hatfield superscalar architecture (HSA) [25]. Instruction traces were then obtained using the HSA instruction-level simulator, with each trace entry providing information on the branch address, branch type and target address. These traces were used to drive a series of trace-driven branch predictors. The trace-driven simulators are highly configurable, the most important parameter being the number of HR bits. As output, the simulators generate the overall prediction accuracy, the number of incorrect target addresses and other useful statistics.

4.2. Conventional two-level predictors

For comparative purposes, we first simulated three global predictors, a GAg predictor, a GAs predictor with 16 PHTs and a GAp predictor (Fig. 2). The average misprediction rate initially falls steadily as a function of global HR length, before flattening out at a misprediction rate of around 9.5%. In general, there is no benefit in increasing the HR length beyond 16 bits for the GAg predictor and 14 bits for the GAs/GAp predictors. Beyond this point there is either no significant benefit from new correlations or any benefit is negated by the additional number of initialisations required in the PHTs.

We also simulated three local predictors: a PAg, a PAs and a PAp predictor (Fig. 3). The local predictors achieve misprediction rates of around 7.5%, significantly better than the global predictors. The best performance of 7.48% is achieved with a PAp predictor and a 28-bit HR. This improvement is largely achieved because local predictors, unlike their global counterparts, continue to benefit from additional HR bits. However, with both global and local two-level predictors the accuracy does not improve smoothly as a function of HR length.

4.3. An LVQ branch predictor

Global predictors perform better with some branches, while local predictors perform better with others. It is therefore highly desirable to combine both forms of prediction within a single predictor. Unfortunately, combining both global and local history registers within a conventional two-level predictor is very costly since both the size and cost of the PHTs increase exponentially as a function of history register length. In contrast, it is very easy to combine global and local history information in a neural predictor since there is no explosive cost increase corresponding to the exponential growth of the PHT size.

Three LVQ predictors were considered with the following inputs:

- PC + HRl
- PC + HRg
- PC + HRl + HRg

The input vector for the neural network was constructed by concatenating the 10 least signifi-

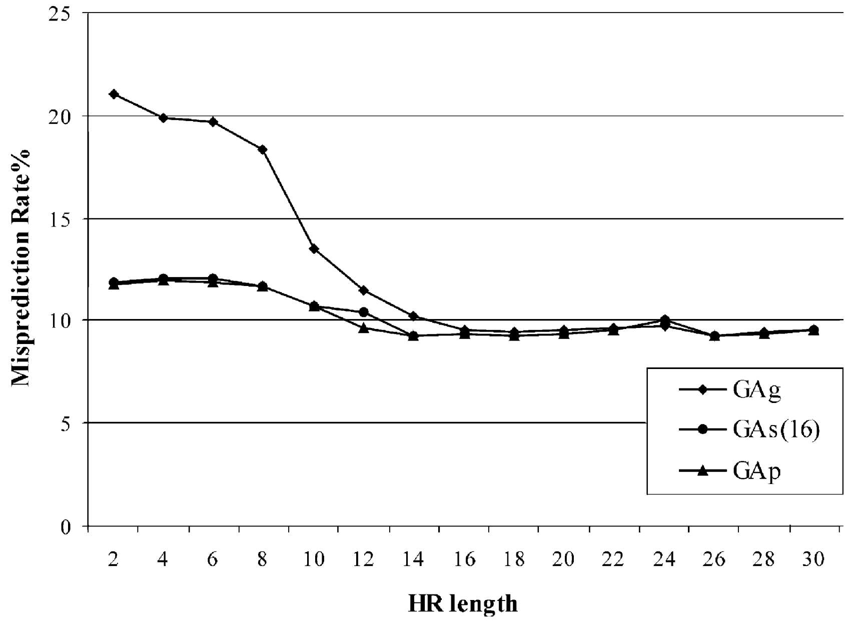

Fig. 2. Conventional global two-level predictors.

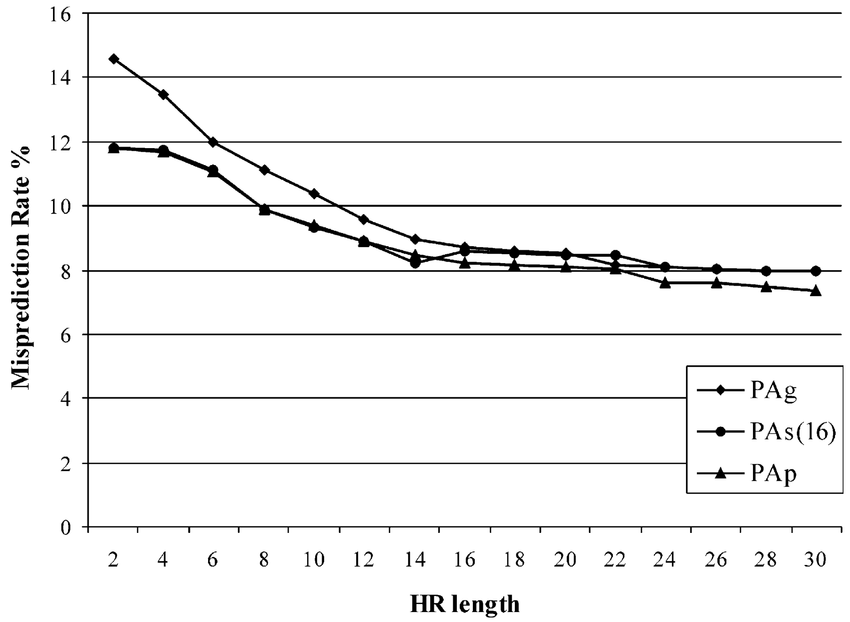

Fig. 3. Conventional local two-level predictors.

cant bits of the PC with HRg, HRl or a combination of HRg + HRl as appropriate. Initially the values of the learning step $a(t)$ were varied between 0.1 and 0.001. Eventually the value $a(t) = 0.01$ was standardised after it had been demonstrated that the predictor was largely insensitive to

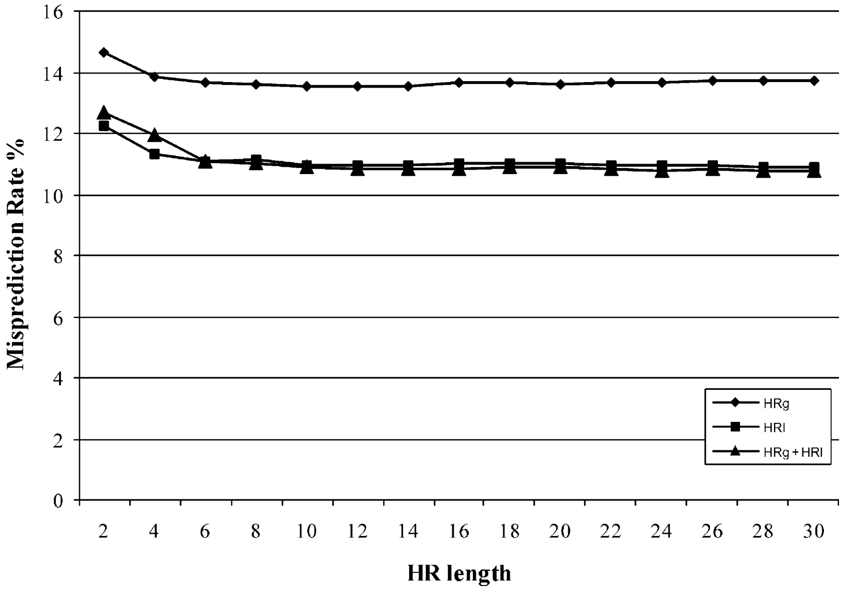

Fig. 4. LVQ branch predictors.

slight variations in $a(t)$. The simulation results are presented in Fig. 4.

The global LVQ predictor achieved an average misprediction rate of 13.54%. Furthermore, only modest further benefits were realised by increasing HR beyond four bits. The global predictor was therefore unable to benefit from large amounts of HRg information. The local LVQ predictor achieved a significantly lower misprediction rate of 10.91%. However, although this figure was recorded with a 30-bit HRl, the local predictor was also unable to adapt to a large amount of history register information, and no significant improvements were observed with HR sizes greater than 6–10 bits. The superior performance of the local predictor is not entirely unexpected; there is likely to be far more positive re-enforcement of prediction information between distinct branches in a local predictor. In contrast, a global predictor must "learn" about each branch separately.

Finally, a hybrid global and local LVQ predictor, with equal numbers of HRg and HRl bits, yielded a marginal improvement on the local LVQ predictor levels, pushing the best average misprediction rate down to 10.78% (see Fig. 4).

Overall, the results of the LVQ predictor are in line with the average accuracy of 88.10% achieved by a classic BTC with these benchmarks. However, global conventional two-level predictors and local conventional two-level predictors achieve superior performance to our LVQ predictors. Our simple LVQ predictors are therefore unable to compete with conventional two-level adaptive predictors. Nevertheless, we found these first results very encouraging. An LVQ network solves a binary classification problem by attempting to find a single multi-dimensional plane that divides the input space, in this case a taken and a not-taken space, into two. Although the plane can be changed dynamically as each branch executes, it appears unlikely that an entirely satisfactory solution can be found. Since our LVQ predictors nonetheless managed to equal the performance of a classic BTC, we were encouraged to develop further neural predictors.

4.4. A backpropagation neural predictor

A total of four backpropagation neural predictors were simulated:

- A global backpropagation neural predictor, using PC + HRg, with binary inputs.

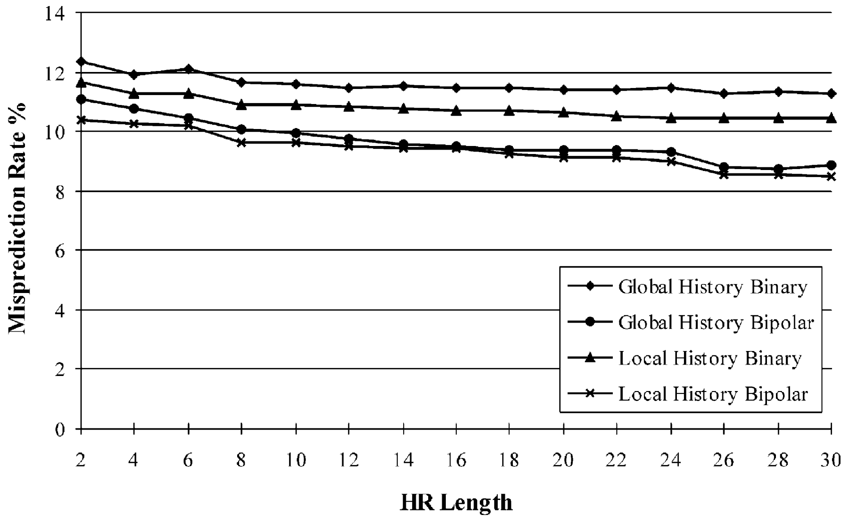

Fig. 5. Backpropagation neural predictors.

- A global backpropagation neural predictor, using PC + HRg, with bipolar inputs.
- A local backpropagation neural predictor, using PC + HRl, with binary inputs.
- A local backpropagation neural predictor, using PC + HRg, with bipolar inputs.

We did not simulate a backpropagation neural predictor that uses a combination of HRg and HRl. The simulation results are plotted in Fig. 5 as a function of HR length. A learning rate of 0.125 was used throughout.

The global backpropagation neural predictor with binary inputs (0 or 1) achieves a misprediction rate of 11.28%, which is significantly better than the global LVQ predictor. However the global backpropagation predictor with bipolar inputs (−1, +1) significantly improves on this figure and achieves a misprediction rate of 8.77%. Intuitively, feeding in not-taken results as minus one allows the backpropagation neural predictor to exploit correlations between not-taken branches and the branch being predicted. In contrast, if not-taken results are fed in as zero, they can have little direct result on the final outcome, since their weighted input into each intermediate neural cell must always be zero. Interestingly, the prediction accuracy also continues to improve as HR length is increased, and only finally dips below a misprediction rate of 9% with a HR length of 26 bits.

The local backpropagation neural predictor consistently outperforms the global backpropagation neural predictor. With binary inputs, the best misprediction rate is 10.46%, while with bipolar inputs 8.47% is achieved. Significantly, backpropagation neural prediction performance is now comparable with the performance of conventional two-level adaptive predictors (see Fig. 6). The best global backpropagation neural predictor with a misprediction rate of 8.77% is 5.2% better than the best GAs predictor, while the best local backpropagation predictor at 8.47% is 13.2% worse than the best PAs predictor (see Fig. 6).

5. Conclusions

In this study, we sought to determine whether a neural network could mimic a two-level adaptive branch predictor and achieve comparable success rates. Two types of neural predictors were simulated, an LVQ predictor and a backpropagation predictor. While the LVQ predictor only achieved results comparable to a traditional BTC, the backpropagation predictor performance was comparable to conventional two-level adaptive predictors. In the case of global predictors, the best

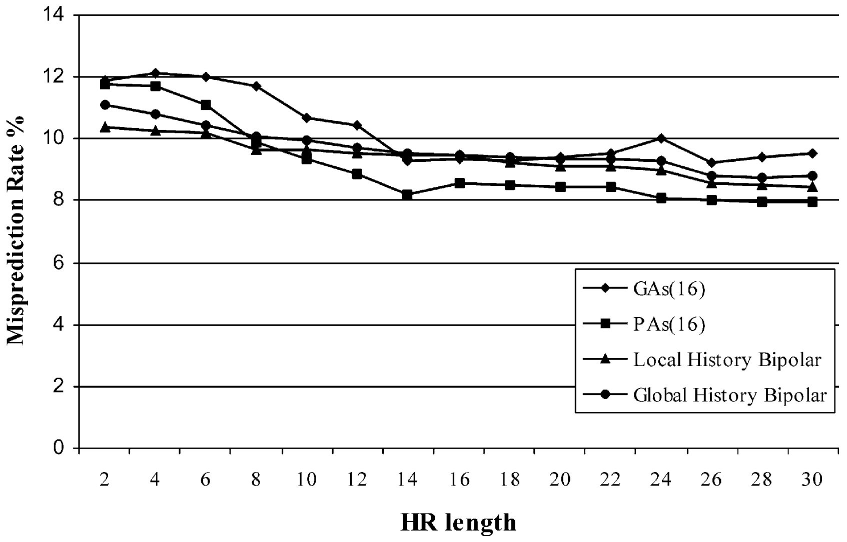

Fig. 6. Conventional two-level predictors compared with backpropagation neural predictors with bipolar inputs.

neural predictor was marginally superior to the best conventional two-level predictor. In the case of local predictors, the best conventional two-level predictor was marginally superior to the best neural predictor. These results suggest that not only can NNs generate respectable prediction results, but in some circumstances a neural predictor may be able to exploit correlation information more effectively than a conventional predictor.

Traditionally, NNs undergo exhaustive and often very time-consuming training before they are used. In this respect, branch prediction appears to be an unpromising application for NNs. In branch prediction, a neural network is expected to sample the outcome of a specific branch once and to then predict the same branch when it is encountered for a second time. The most exciting result of these simulations is therefore the extent to which backpropagation NNs are able to assimilate and benefit from large amounts of history register information with a minimum of training. The distinct drop in the bipolar backpropagation misprediction rate when 26 bits of HR are used is a good illustration of this result. Our results therefore suggest that NNs can adapt rapidly enough to be successfully used in dynamic branch prediction.

NNs provide an extremely interesting topic for future branch prediction research. One challenge is to construct composite input vectors for neural network predictors that will enable them to outperform conventional predictors. This task involves both identifying new correlation mechanisms that can be exploited by neural prediction and tailoring the input information to fully exploit the capabilities of a neural predictor.

Work is ongoing to design hardware implementations of the neural models simulated, and it seems feasible to build neural network predictors that work quickly enough to be useful within a sensible silicon budget. For example, by coding the $+1/-1$ signals as 0/1 and the weights and thresholds as small fixed-point signed fractions, the apparently expensive multiplications and floating-point arithmetic become much simpler and faster operations.

References

[1] T. Ball, J. Larus, Branch Prediction for Free, Proceedings of the SigPlan93 Conference on Programming Language and Implementation, June 1993, pp. 300–313.
[2] D. Burger, J.R. Goodman, Billion-transistor architectures, IEEE Computer (September) (1997) 46–49.
[3] B. Calder, D. Grunwald, D. Lindsay, Corpus-based static branch prediction, SIGPLAN Notices, June 1995, pp. 79–92.

[4] R. Callan, The Essence of Neural Networks, Prentice-Hall, 1999.

[5] P. Chang, E. Hao, T. Yeh, Y. Patt, Branch Classification: A New Mechanism for Improving Branch Predictor Performance, Micro-27, San Jose, California, November 1994, pp. 22–31.

[6] P. Chang, E. Hao, Y.N. Patt, Alternative Implementations of Hybrid Branch Predictors, Micro-29, Ann Arbor, Michigan, November 1995, pp. 252–257.

[7] I.K. Chen, J.T. Coffey, T. Mudge, Analysis of Branch Prediction via Data Compression, Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII), Cambridge, MA, USA, October 1996, pp. 128–137.

[8] C. Egan, Dynamic Branch Prediction in High Performance Superscalar Processors, PhD thesis, University of Hertfordshire, August 2000.

[9] C. Egan, G.B. Steven, W. Shim, L. Vintan, Applying caching to two-level adaptive branch prediction, in: Digital Systems Design Architectures, Methods and Tools, Warsaw, Poland, September 2001, pp. 186–193.

[10] C. Egan, G.B. Steven, L. Vintan, Cached Two-level Adaptive Branch Predictors with Multiple Stages, Lecture Notes in Computer Science (LNCS) Trends in Network and Pervasive Computing (ARCS-2002), Karlsruhe, Germany, April 2002, pp. 179–191.

[11] M. Evers, S.J. Patel, R.S. Chappell, Y.N. Patt, An Analysis of Correlation and Predictability: What makes Two-level Branch Predictors Work, ISCA '25, Barcelona, Spain, June 1998, pp. 52–61.

[12] S.I. Gallant, Neural Networks and Expert Systems, MIT Press, 1993.

[13] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, third ed., Morgan Kaufmann Publishers, 2002.

[14] D.A. Jiménez, C. Lin, Dynamic Branch Prediction with Perceptrons. Proc. of the 7th International Symposium on High Performance Computer Architecture (HPCA-7), Monterrey, NL, Mexico 2001, pp. 197–296.

[15] Intel Corporation. A Tour of the Pentium® Pro Processor Microarchitecure. http://www.intel.com/procs/p6/p6white/p6white.htm.

[16] R.E. Kessler, The Alpha 21264 Microprocessor, IEEE Micro (March) (1999) 24–36.

[17] C.C. Lee, I.-C.K. Chen, T.N. Mudge, The Bi-Mode Branch Predictor, Micro-30, Research Triangle Park, North Carolina, December 1997, pp. 4–13.

[18] S. McFarling, Combining Branch Predictors, WRL Technical Note, TN36, DEC, June 1993.

[19] T.N. Mudge, I. Chen, J. Coffey, Limits of Branch Prediction, Technical Report, Electrical Engineering and Computer Science Department, The University of Michigan, Ann Arbor, Michigan, USA, January 1996.

[20] R. Nair, Dynamic Path-Base Branch Correlation, Micro-28, Ann Arbor, Michigan, November 1995, pp. 15–23.

[21] S. Pan, K. So, J.T. Rahmeh, Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation, ASPLOS-V, Boston, October 1992, pp. 76–84.

[22] Y.N. Patt, S.J. Patel, D.H. Friendly, J. Stark, One billion transistors, one uniprocessor, one chip, IEEE Computer 1 (September) (1997) 51–57.

[23] S. Sechrest, C. Lee, T. Mudge, The Role of Adaptivity in Two-Level Branch Prediction, Micro-29, Ann Arbor, Michigan, November 1995, pp. 264–269.

[24] E. Sprangle, R.S. Chappell, M. Alsup, Y.N. Patt, The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference, ISCA'24, Denver, Colorado, June 1997, pp. 284–291.

[25] G.B. Steven, D.B. Christianson, R. Collins, R. Potter, F.L. Steven, A superscalar architecture to exploit instruction level parallelism, Microprocessors and Microsystems 20 (7) (1997) 391–400.

[26] G.B. Steven, C. Egan, P. Quick, L. Vintan, Reducing Cold Start Mispredictions in Two-level Adaptive Branch Predictors, CSCS-12, Bucharest, Romania, May 1999, pp. 145–150.

[27] G.B. Steven, C. Egan, L. Vintan, A Cost Effective Cached Correlated Two-level Adaptive Branch Predictor, 18th IASTED International Conference in Applied Informatics, Innsbruck, Austria, February 2000.

[28] L.N. Vintan, C. Egan, Extending Correlation in Branch Prediction Schemes, Euromicro99, Vol. 1, Milan, Italy, September 1999, pp. 441–448.

[29] L. Vintan, Predicting Branches through Neural Networks: an LVQ and an MLP Approach, Technical Report University of "Lucian Blaga", Sibiu, Romania, February 1999.

[30] L. Vintan, M. Iridon, Towards a High Performance Neural Branch Predictor, International Joint Conference on Neural Networks, Washington, USA, July 1999.

[31] L. Vintan, Towards a Powerful Dynamic Branch Predictor, Romanian Journal of Information Science and Technology, Bucharest, Romania, 2000.

[32] G. Steven, C. Egan, R. Anguera, L. Vintan, Dynamic Branch Prediction using Neural Networks, Proceedings of International Euromicro Conference DSD'2001, Warsaw, Poland, September, 2001, pp. 178–185, ISBN 0-7695-1239-9.

[33] T. Yeh, Y.N. Patt, Two-Levels Adaptive Training Branch Prediction, Micro-24, Albuquerque, New Mexico, November 1991, pp. 51–61.

[34] T. Yeh, Y. Patt, Alternative Implementations of Two-Level Adaptive Branch Prediction, ISCA -19, Gold Coast, Australia, May 1992, pp. 124–134.

[35] T. Yeh, Y. Patt, A Comprehensive Instruction Fetch Mechanism for a Processor Supporting Speculative Execution, Micro-25, Portland, Oregon, December 1992, pp. 129–139.

[36] T. Yeh, Y.N. Patt, A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History, ISCA-20, San Diego, May 1993, pp. 257–266.

Colin Egan received his BSc degree in Computer Science (Systems Engineering) in 1996 and his PhD in Computer Architecture for his work in Dynamic Branch Prediction in High Performance Superscalar Processors in 2000, from the University of Hertfordshire. He is currently a Senior Lecturer in the Department of Computer Science at the University of Hertfordshire teaching on a wide range of Computer Systems and Computer Architecture courses. He has active research interests in the fields of High Performance Processor Design and Dynamic Branch Prediction.

Gordon Steven graduated from Princeton University, USA with a BSE (High Honors, Phi Beta Kappa) in Electronics. He received his MSE in Electronics in 1967, also from Princeton, and his PhD in Computer Engineering from Manchester University for his work on MU5. Following university Gordon Steven spent 10 years in industry developing various computer systems. At Plessey he worked on the development of the PP250 capabilities based multiprocessor system, at CTL he worked on the Modula-1 minicomputer and at HSDE he worked on early microprocessor based control systems. In 1979, Gordon Steven joined the Computer Science Department of the University of Hertfordshire where he was a Reader in Computer Architecture until his retirement in September 2002.

Patrick Quick graduated from the University of Cambridge in 1973 with a B.A. in Mathematics. After teaching Mathematics and then Computer Science for some years, he obtained a Post-Graduate Diploma in Computer Science in 1987, again from the University of Cambridge. Since 1987 he has been a Senior Lecturer in Computer Science at the University of Hertfordshire teaching on a wide range of Computer Systems and Programming courses. He has active research interests in the fields of High Performance Processor Design and Dynamic Branch Prediction.

Rubén Anguera completed his undergraduate studies in Computer Science at the University of Hertfordshire in 2000. His final year project was entitled ‘Using Neural Networks to Perform Dynamic Branch Prediction in a High Performance Superscalar Architecture’. Some of the results of his project have been used in this paper.

Fleur Steven graduated from the Hull University in 1985 with a B.Sc. in Zoology. She received both her M.Sc. in 1986 and her PhD for her work in Computer Architecture in 1989, from the University of Hertfordshire. She was a post-doctoral research fellow at the University of Hertfordshire until April 2001.

Lucian Vintan obtained an MSE (Computer Science) in 1987 and a PhD (Computer Science) in 1997 from the University “Politechnica” of Timisoara, Romania. He is a Professor of Computer Science and Engineering at the University “Lucian Blaga” of Sibiu, Sibiu, Romania. Lucian is an active researcher in the fields of High Performance Processor Design and Dynamic Branch Prediction and has collaborated with the Computer Architecture Research Group at the University of Hertfordshire since 1996.