***Towards a High Performance Neural Branch Predictor*, Proceedings of The International Joint Conference on Neural Networks - IJCNN '99 (CD-ROM, ISBN 0-7803-5532-6), Washington DC, USA, 10-16 July, 1999**

# Towards a High Performance Neural Branch Predictor

*Lucian N. VINTAN*
*University "L. Blaga", Dept. of Comp. Sc.*
*Str. E. Cioran, No. 4, Sibiu-2400, ROMANIA*
*Tel./fax:++40-69-212716*
*E-mail:* lucian.vintan@ulbsibiu.ro

**Abstract:** *The main aim of this short paper is to propose a new branch prediction approach called by us "neural branch prediction". We developed a first neural predictor model based on a simple neural learning algorithm, known as Learning Vector Quantization algorithm. Based on a trace driven simulation method we investigated the influences of the learning step and training processes. Also we compared the neural predictor with a powerful classical predictor and we establish that they result in close performances. Therefore, we conclude that in the nearest future it might be necessary to model and simulate other more powerful neural adaptive predictors, based on more complex neural networks architectures or even time series concepts, in order to obtain better prediction accuracies compared with the previous known schemes.*

**Key Words:** *MII Architectures, Branch prediction,Trace driven simulation, Neural algorithms (LVQ, MLP)*

## 1. INTRODUCTION

As the average instruction issue rate and depth of the pipeline in multiple instruction issue (MII) processors increase, the necessity of an efficient hardware branch predictor becomes essential. Very high prediction accuracies are necessary, because taking into account the MII processors characteristics as pipeline depth or issue rates, even a prediction miss rate of a few percent involves a substantial performance loss.

The main aim of this work is to propose a new branch prediction approach called neural branch prediction. Our work hypothesis will consider branch prediction as a particular problem belonging to pattern recognition class and therefore, we consider it's desirable to use neural networks in order to predict branches. Also, we investigate comparatively, through a trace driven simulation method, a classical branch prediction scheme proceeded from Professor Yale Patt's research group [Yeh92, Cha97, Eve96] with some modifications and the proposed neural branch predictor, both of them integrated into a MII environment. We used the traces obtained based on the eight C Stanford integer benchmarks. These benchmarks were compiled through the HSA (Hatfield Superscalar Architecture) compiler, developed at the University of Hertfordshire, UK, by Dr. G.B. Steven's research group. Further, the traces were obtained using the HSA simulator, developed at the same university [Ste96]. Based on these tools, we have developed an original simulator to investigate some branch prediction schemes.

The first efficient approach in hardware branch prediction consists in Branch Target Buffer (BTB) structures [Per93]. BTB is a small associative memory, integrated on chip, that retains the addresses of recently executed branches, their targets and optionally other information (e.g. target opcode). Due to some intrinsic limitations, BTB's accuracies are limited on some benchmarks having unpropitious characteristics (e.g. correlated branches).

In order to improve BTB's efficiency, Yeh and Patt (1992) generalised it through a new approach called Two Level Adaptive Branch Prediction. According to [Yeh92], the Two Level Adaptive Branch Prediction uses two distinct levels of branch history information to make predictions. The first level consists in the History Register (HR), that contains the last k branches encountered (taken/ not taken) or the last k occurrences of the same branch instruction. The second level consists in the branch behaviour of the last l occurrences of the specific pattern of these branches. It is implemented by a Pattern History Table (PHT), that contains essentially the branch prediction automaton ( usually 2 bit saturating counters).

HR shifts left with a binary position when updated according to the actual branch behaviour (taken=1/ not taken=0). There is a corresponding entry in the PHT for each of the $2^k$ HR's patterns.

The prediction of the branch (P) is a function (f) of the actual prediction automaton state $S_t$.

$$\mathbf{P = f(S_t)} \qquad \mathbf{(1)}$$

After the branch is resolved, HR is shifted left and the prediction automaton state becomes $S_{t+1}$.

$$\mathbf{S_{t+1} = g( S_t ,B_t)} \qquad \mathbf{(2)}$$

where g represents the automaton's transition function and $B_t$ represents the behaviour of the last branch encountered (taken/ not taken).

These Two Level Adaptive Branch Prediction schemes are very effective in predicting correlated branches with high accuracy. It's well known that the average prediction rate for these schemes, measured on nine of the ten SPEC benchmarks, is about 97%, while BTB schemes achieved at most 94% on the same benchmarks [Yeh92]. An excellent generalisation of these Two Level Adaptive Branch Prediction schemes, based on the universal compression/prediction algorithm called "prediction by partial matching", is given in [Mud96].

Further, we'll try to propose a new distinct branch prediction approach, based on some pattern recognition concepts like neural networks, taking into account their real adaptive behaviour in other similar problems. Thus, we'll look at branch prediction as a pattern recognition problem.

## 2. A MODIFIED PAp PREDICTOR (MPAp)

In order to offer a classical equivalence for the proposed neural predictor, we propose a Two Level Adaptive Branch Predictor derived - with some modifications - from a PAp (Per Address Branch History Table and Per Address Pattern History Tables) scheme, first presented in

[Yeh92]. The prediction process through a MPAp scheme is based on three orthogonal information: branch's PC least significant bits (noted with PCl, on i bits), a global history register (noted with HRg) containing the last k branches encountered and a set of history registers per each branch (HRl) containing the last l occurrences (taken/not taken) of the same branch instruction (branch history). The original PAp scheme uses for the prediction process only PC low and HRl information, therefore it neglects the possible correlation with other branches (HRg information). Also, in the original PAp scheme, the PBHT (Per Address Branch History Table) table contains no tags. As it can be noticed in figure 1, the MPAp scheme contains two tables: PBHT and PPHT (Per Address Pattern History Tables).

Therefore, in the MPAp scheme I use both a global HR and a per branch HR to make predictions. Also, MPAp provide a per branch history register for every value of the global history register (HRg). Neglecting HRg (HRg on 0 bits) a classical PAp scheme (with an additional Tag stored in PBHT) it's obtained, while neglecting HRl fields a classical GAp scheme it's obtained - first presented in [Pan92] - having an additional Tag too. Unfortunately, as it will be shown further in paragraph 4.2, using both HRg and HRl seems not to be an advantage over HRl alone (PAp scheme).

Our PBHT contains $2^{i+k}$ sets, and each set contains $2^j$ associative entries with a LRU (Least Recently Used) evacuation algorithm. Our PPHT contains $2^{i+k+l}$ entries, each entry containing a prediction automaton (2 bits saturating counter) and the branch's target address. In our opinion MPAp could be a complex powerful prediction scheme, that will be compared further to the new neural proposed predictor described in the next paragraph.

## 3. THE LVQ NEURAL BRANCH PREDICTOR

We propose a new branch prediction method, based on an real adaptive algorithm named Learning Vector Quantization [Koh95], belonging to neural network (NN) algorithms. Therefore, we'll adapt the Learning Vector Quantization (LVQ) algorithm to the branch prediction problem. From the large class of NN algorithms [Gal93] we chose LVQ algorithm because it is one of the simplest NN algorithms and suited to the branch prediction problem as it can be proved further.

As in the Two Level Adaptive Branch Prediction, in this case the run-time prediction process is based on the same three orthogonal information: the branch's PC low (PCl, on i bits), the history of the k previous branches named HRg (Global History Register on k bits) and the branch's own history (taken/not taken) named HRl (Local History Register, on l bits). Also, similarly to the Two Level Adaptive Branch Prediction, the LVQ predictor will use all these information together with the branch's target address from tables like those presented in our modified PAp scheme (Figure 1). In contrast, this time it's not necessary to implement the classical $2^{i+k+l}$ prediction automata stored in PPHT table. These automata will be replaced with a single global LVQ (neural) prediction structure, as it will be described further.

The LVQ prediction structure contains two binary vectors ("codebook vectors") named Vnt and Vt, each of them on (i+k+l) bits, associated with the "not taken" event, respectively with the "taken" event. Initially, Vnt will be "all zero" and Vt will be "all one". During the prediction process the vector X is defined, associated with the current branch to be predicted. It contains the branch's PCl concatenated with the branch's HRg and HRl fields. If we note S={Vnt, Vt} then we can define the "winner vector" (Vw) as that vector belonging to S, that has a minimum Hamming distance (HD) to vector X ( **HD=** $\sum_{p=1}^{i+k+l}(X_p - Vw_p)^2$). If Vw = Vnt the prediction will be "not taken" and if Vw = Vt then the prediction will be "taken". The other vector belonging to S will be named the "loser vector" (Vl).

After the branch's output is known, the two vectors belonging to S, will be modified according to the following relations:

$$\mathbf{Vw(t+1) = Vw(t) \pm a(t)[X(t) - Vw(t)]} \quad \mathbf{(3)}$$

("+" for a correct prediction and "-" for an incorrect prediction) and:

$$\mathbf{Vl(t+1) = Vl(t)} \quad \mathbf{(4)}$$

where a(t) represents the "learning step"; it may be constant as in the classical LVQ algorithm (recommended to be smaller then 0.1) or decrease monotonically with time like in the "optimized LVQ algorithm". In the latter case, a(t) is given by the following recurrent equation [Koh95]:

$$\mathbf{a(t+1) = a(t) / (1 \pm a(t))} \qquad \mathbf{(5)}$$

(Also, "+" for a correct prediction and "-" for an incorrect prediction). In this case it's necessary that a(t) be smaller than the initial step. Theoretically it can be possible, but simulations show that is not efficient, that the loser vector to be also modified ("discouraged") in equation (4).

Based on the adaptive heuristic nature of the LVQ prediction algorithm, the Vnt vector will tend dynamically to a "not taken" pattern (class), while Vt vector will tend to a "taken" pattern (class). The predictor will learn continuously, therefore the adequate class - more alike with the incoming X(t) binary pattern - will "attract" with more and more accuracy the newly income X(t) vector, involving thus better predictions. Each classic branch prediction scheme has a corresponding LVQ branch prediction scheme that uses similar tables for storing HRg, HRl and target addresses (PBHT and PPHT in this case). Through this method, we approach the branch prediction problem as a pattern recognition problem, where the pattern X must be recognised belonging exclusively to one of two possible classes (Vt-Taken /Vnt- Not taken). Of course, some adequate training algorithms are further necessary, in order to obtain better accuracies.

Therefore, through this approach we establish a possible link between branch prediction problems and respectively pattern recognition problems solved with NN. Of course, it's possible to extend these ideas to other neural branch predictors also having X(t) vector as an unique input and as output the prediction itself (one bit). For example, a multilayer perceptron with a backpropagation learning algorithm [Gal93] could be a powerful and feasible branch predictor, also adaptive.

Another interesting neural predictor feature could be related to the target prediction for indirect jumps. Taking into account that the target of an indirect branch can change with every dynamic instance of that branch, predicting its target is really difficult. There are few solutions to this problem and all of them involve a great deal of further work. One of the most recent valuable approaches in this sense, improves the indirect jumps prediction accuracy by choosing its target from the most recent targets of the indirect jump that have already been encountered [Cha97], based on a simply heuristic. In contrast, our new approach proposes that the NN that predicts the branch direction also predict the target effective address, for indirect jumps only, based on the same information: PC, HRg and HRl (Figure 2). Therefore, this approach is based on a supposed correspondence between the indirect jump's dynamic patterns (PC, HRg, HRl) and its dynamic target addresses. In the nearest future, based on a trace driven simulation method, we'll quantitatively investigate this approach related to target prediction for indirect jumps.

The difficult problem related to these neural predictors is to establish whether they can be implemented on a chip, taking into account the run-time prediction request. More precisely, that means the prediction must be done during the instruction fetch phase for an efficient approach. Based on the present technological progresses, that allow performant NN hardware implementations, in our opinion the neural predictor idea could be feasible and, therefore, new investigations in this research area are warranted. At this time, our intuition is that a simplified NN predictor could be designed within the timing restraints of a superscalar. We also suspect that the cost would be far less than one of Two Level Adaptive predictors and it may even be possible to implement multiple cut-down NN predictors, associated which each branch. Anyway, NN predictors could be a useful approach in establishing, estimating and understanding better, the processes of branch predictability. In fact, the neural predictor replaces the $2^{i+k+l}$ prediction automata with one global neural prediction structure as we described above.

## 4. SIMULATION WORK

### 4.1. BENCHMARKS PROGRAMS

The simulation work has been centred on the Stanford integer benchmark suite, a collection of eight C programs designed by Professor John Hennessy, to be representative of non - numeric code while at the same time being compact. The benchmarks are computationally intensive with higher dynamic instruction counts. All these benchmarks were compiled by the HSA gnu C compiler which targets the HSA instruction set. The resulted HSA object code was simulated by a dedicated HSA simulator [Ste96], that generates the corresponding traces. Some characteristics of the used traces are given in Table 1.

The average instructions number is about 273.000 and the average percentage of total instructions that are branches is about 18%, with about 76% of them being taken. Derived from HSA traces, special traces were obtained, containing exclusively all the processed branches.

Each branch belonging to these modified HSA traces is stored in the following format: branch's type, the PC of the branch and it's target address.

Following our aims, we developed a dedicated trace driven simulator that uses the above mentioned traces. The most important input parameters for this simulator are:

- the number of PCl bits (i), HRg bits (k) and HRl bits (l)
- number of associative entries belonging to a PBHT set ( $2^j$ associative entries per set)
- the learning step value ( for the LVQ neural predictor only)

Taking into account Stanford benchmark's characteristics together with the present technological on-chip integration level, during the simulation PBHT and PPHT tables up to 512 entries were considered.

## 4.2. SOME RESULTS

Table 2 presents in a comparative manner, some results for a MPAp scheme having i=2, j=3, k=6, l=0 (therefore a GAp scheme) respectively a MPAp scheme with i=2, j=3, k=0, l=6 (a classical PAp scheme with Tag in PBHT). As it can be noticed, PAp scheme outperforms GAp with an average prediction accuracy of 90.08% compared to 89.59% for GAp. Also, surprisingly, as it can be seen comparing Table 2 with Table 5, this PAp scheme outperforms with 0.32% an equivalent MPAp scheme (i=2, j=3, k=3, l=3, having thus the same PBHT & PPHT capacities as the PAp scheme), characterised by an average prediction accuracy of 89.76%. As another surprising example, a PAp with i=2, j=3, l=5 and k=0 obtains an average accuracy of 89.87%, greater than 88.6%, the average accuracy for an equivalent MPAp with i=2, j=3, l=3 and k=2 (see Table 4). Because of the additional warm up time the MPAp model is likely to perform less successfully with short benchmarks like the Stanford benchmarks. However, we would expect our new MPAp configuration to show some improvements with Spec benchmarks which are larger.

Table 3 shows the prediction accuracies for different learning step values (a), for a neural LVQ predictor having the parameters: i=2, j=3, k=3 and l=3. According to other similar simulations, the optimal step seems to be a=0.01, that involves an average arithmetic accuracy of 88.37% in this case. Therefore, in all the further analysis presented here we'll consider the learning step a=0.01.

For a variable learning step corresponding to an "optimized LVQ", according to equation (5), simulations point out surprisingly, average prediction accuracies lower with about 1.5%, compared to previous constant learning steps. Also - based on a large set of simulations - the LVQ's predictor dynamically training process, involves an average accuracy growth of only about 0.5%. That means the average accuracy is with about 0.5% greater after processing the same benchmark again, of course without LVQ structures initialisations.

Tables 4 to 11 present some comparisons between a classic MPAp predictor and the corresponding LVQ neural predictor, for different realistic i, j, k and l parameters. As it can be seen, the classical predictor involves average accuracies better with 0.19% to 1.4% in all these cases. Tables 4 to 7 show that varying the prediction tables capacities in reasonable limits according to Stanford benchmarks characteristics, for both MPAp and LVQ schemes, the optimal parameters seems to be i=j=k=3 and l=4 (see table 7). Growing i from 3 to 5, didn't improve performances (see table 8). Tables 9 and 10 shows clearly that k growth involves better accuracies for the LVQ predictor, practically at the same level with the MPAp predictor now. Unfortunately, in both these cases the prediction tables are too huge. Finally, table 11 shows that growing the branch's "local history", the MPAp's accuracy grows too. However surprising, on "matrix" and "sort" (a benchmark well known as very difficult predictable) benchmarks, the LVQ predictor obtains slightly better prediction accuracies. We could appreciate these first results as optimistic, taking into account that the LVQ structure and its prediction algorithm are quite simple and therefore not very performant. Even so, it involves accuracies comparable to the MPAp predictor. We believe that more powerful NN architectures also together with more powerful learning algorithms could produce significantly better prediction accuracies.

## 5. CONCLUSIONS AND FURTHER WORK

We proposed a new branch prediction approach based on NN concepts, called by us neural branch prediction. More precisely, we modelled here a neural predictor characterised by a very simple adaptive learning algorithm named LVQ. Based on a trace driven simulation method we investigated the influences of the learning step values and training processes. Also we compared the neural LVQ predictor with a powerful classical predictor and we establish that they involve quite close performances, which is encouraging from our point of view.

All contemporary branch prediction techniques are based essentially on FSM (Finite State Machine) prediction automata. As an alternative, our approach replaces all the FSM prediction automata involved with a simple NN.

Therefore, we conclude that in the nearest future it is interesting to model and simulate other **more powerful neural predictors,** based on more complex NN or even time series concepts (perhaps another interesting challenge!), in order to obtain better prediction accuracies compared to the previous known schemes. Also, we'll use NN in order to deal with predicting targets for indirect jumps, an open problem at this moment.

In this sense, as a first further investigation, we believe that a more powerful neural branch predictor could consist of a **multilayer perceptron** (MLP) or even a recurent NN, with a **backpropagation learning algorithm** [Gal93]. As a first step a MLP with one hidden (intermediate) layer is proposed. The input layer will contain (i+k+l) cells, corresponding to the three prediction information (PCl, HRg, HRl). The intermediate layer will contain a parameterised number of cells, usually greater than the input layer. We suppose that this MLP approach could be useful even as a metric of the inherent limit of predictability of applications themselves.

A common criticism for all the present Two Level Adaptive Branch Prediction schemes consists in the fact that they used **an insufficient global correlation information** (HRg). If each bit belonging to HRg will be associated with **its corresponding PC** during the prediction process, the correlation information will be more complete (even compressed through hashing!) and therefore the prediction accuracy would be better. In this way it will be not only known if the previous k encountered branches were taken or not (through HRg content), but it will be exactly known **which branches they were ($B_1$ $B_2$ ... $B_k$ ).** Another set of open problems could be represented by the following question: how could be correctly predicted a branch that leave a loop (thus non taken), of course, the number of loop's iterations is considered unknown (arbitrarily). These problems are very important, and we'll try to solve them with a further paper.

## ACKNOWLEDGEMENTS

This work was supported in part by the Romanian Ministry of Research and Technology grant MCT No. 4086/1998 respectively by the Romanian National Council of Academic Research grant CNCSU No. 391/1998.

Also I like to thank **Mr. Macarie Breazu** from the Department of Computer Engineering at the University "Lucian Blaga" of Sibiu, for providing some NN references and for patiently and competently explaining some related concepts.

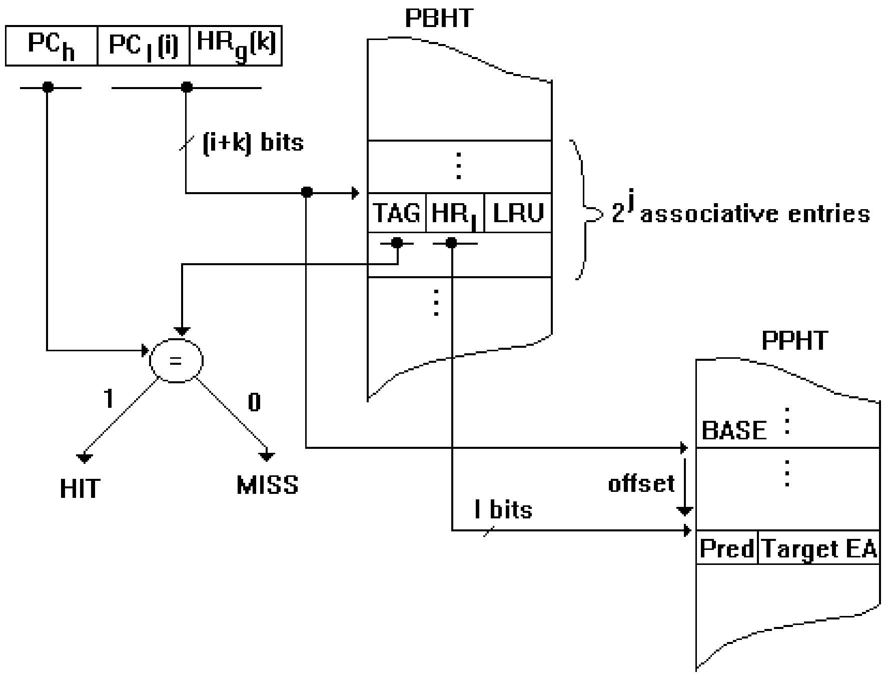

**Figure 1. A modified PAp branch predictor scheme**

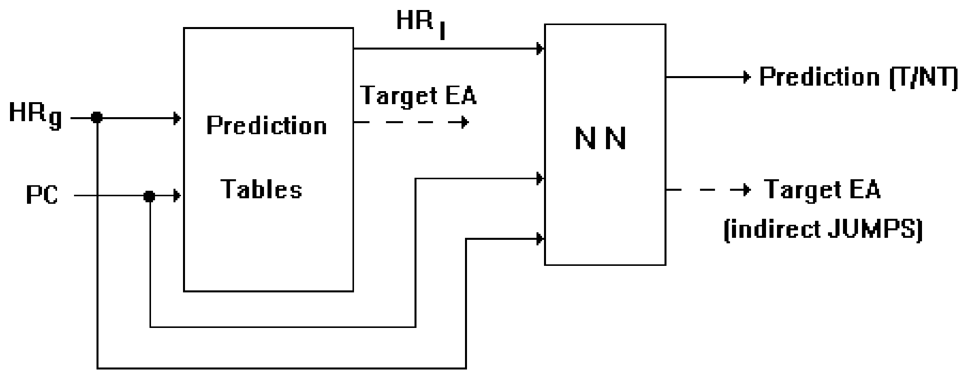

**Figure 2. Block diagram of the neural predictor**

| Benchmark | Total instr. | % Branches (%Taken) | Description |
|---|---|---|---|
| puzzle | 804.620 | 25(91) | Solves a cube packing problem |
| bubble | 206.035 | 20(75) | Bubble sorts an array |
| matrix | 231.814 | 9(97) | Matrix multiplication |
| permute | 355.643 | 15(80) | Recursive computation of permutations |
| queens | 206.420 | 19(50) | Solves the eight queens problem |
| sort | 72.101 | 17(65) | Quick sorts a randomised array |
| towers | 251.149 | 15(76) | Solves Towers of Hanoi problem (recursive) |
| tree | 136.040 | 24(73) | Performs a binary tree sort |

**Table 1. Characteristics of the HSA traces**

| | puzz | bub | matr | perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **GAg** | 95.90 | 84.80 | 96.70 | 95.30 | 81.10 | 77.76 | 95.37 | 89.80 | **89.59** |
| **PAp** | 95.00 | 88.60 | 96.34 | 94.95 | 83.20 | 74.00 | 98.64 | 89.91 | **90.08** |

**Table 2. Prediction accuracies for a GAg respectively a PAp scheme (i=2, j=3 and history registers on 6 bits)**

| a | puzz | bub | matr | perm | queen | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| 0.1 | 94.96 | 85.75 | 96.7 | 88.14 | 78.53 | 77.42 | 95.8 | 89.21 | **88.31** |
| 0.01 | 94.95 | 85.07 | 96.68 | 88.35 | 79.49 | 76.65 | 96.3 | 89.3 | **88.37** |
| 0.001 | 94.96 | 83.77 | 96.63 | 87.63 | 79.14 | 75.22 | 95.88 | 88.97 | **87.77** |

**Table 3. Prediction accuracies for different learning step values (i=2, j=3, k=3, l=3)**

| | puzz | bub | matr | perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **Neuro** | 95.1 | 85.14 | 96.67 | 88.45 | 77.82 | 77.4 | 96.3 | 89.2 | **87.3** |
| **MPAp** | 95.6 | 84.97 | 96.5 | 87.88 | 81.96 | 75 | 97.1 | 89.7 | **88.6** |

**Table 4. Prediction accuracies for a neural branch predictor and a MPAp predictor (i=2, j=3, k=2, l=3)**

| | puzz | bub | matr | perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **Neuro** | 95 | 85.17 | 96.68 | 88.35 | 79.5 | 76.65 | 96.3 | 89.3 | **88.36** |
| **MPAp** | 95.6 | 84.98 | 96.48 | 97.76 | 81.53 | 74.93 | 97.1 | 89.7 | **89.76** |

**Table 5. Prediction accuracies for a neural branch predictor and a MPAp predictor (i=2, j=3, k=3, l=3)**

| | puzz | bub | matr | Perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **Neuro** | 95.19 | 85.3 | 96.68 | 89.66 | 80 | 76.68 | 97 | 89.60 | **88.76** |
| **MPAp** | 95.7 | 86.5 | 96.4 | 98.15 | 82.4 | 74 | 97.6 | 89.60 | **90.03** |

**Table 6. Prediction accuracies for a neural branch predictor and a MPAp predictor (i=2, j=3, k=3, l=4)**

| | puzz | bub | matr | perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **Neuro** | 95.21 | 85.3 | 96.68 | 89.66 | 80 | 76.70 | 97 | 89.6 | **88.77** |
| **MPAp** | 95.68 | 86.5 | 96.40 | 98.20 | 82.38 | 74.1 | 97.6 | 89.66 | **90.06** |

**Table 7. Prediction accuracies for a neural branch predictor and a MPAp predictor (i=3, j=3, k=3, l=4)**

| | puzz | bub | matr | perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **Neuro** | 95.24 | 85.20 | 96.54 | 89.60 | 81.06 | 76.15 | 96.85 | 89.43 | **88.76** |
| **MPAp** | 95.64 | 86.41 | 96.26 | 98.10 | 82.27 | 73.57 | 97.48 | 89.46 | **89.89** |

**Table 8. Prediction accuracies for a neural branch predictor and a MPAp predictor (i=5, j=3, k=3, l=4)**

| | puzz | bub | matr | perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **Neuro** | 95.47 | 84.85 | 96.50 | 94.10 | 81.00 | 76.16 | 96.84 | 89.42 | **89.29** |
| **MPAp** | 95.83 | 89.38 | 96.06 | 97.93 | 81.99 | 71.17 | 98.32 | 89.00 | **89.95** |

**Table 9. Prediction accuracies for a neural branch pre dictor and a MPAp predictor (i=3, j=3, k=6, l=4)**

| | puzz | bub | matr | perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **Neuro** | 95.44 | 85.11 | 96.43 | 95.07 | 82.81 | 75.18 | 96.75 | 89.32 | **89.51** |
| **MPAp** | 96.08 | 89.88 | 95.81 | 98.45 | 83.91 | 66.77 | 98.48 | 88.24 | **89.70** |

**Table 10. Prediction accuracies for a neural branch predictor and a MPAp predictor (i=3, j=3, k=9, l=4)**

| | puzz | bub | matr | perm | queens | sort | tow | tree | AM |
|---|---|---|---|---|---|---|---|---|---|
| **Neuro** | 94.63 | 85.28 | 96.48 | 93.04 | 81.16 | 76.47 | 96.83 | 89.62 | **89.19** |
| **MPAp** | 96.06 | 92.27 | 95.86 | 98.09 | 86.11 | 68.62 | 98.37 | 89.18 | **90.57** |

**Table 11. Prediction accuracies for a neural branch predictor and a MPAp predictor (i=3, j=3, k=3, l=9)**

## REFERENCES

***[Cha97]* Chang P.Y., Hao E., Patt Y.N**. - *Target Prediction for Indirect Jumps*, ISCA '97 - Ann. Int.'L Symp. Computer Architecture (http://www.eecs.umich.edu/HPS)

***[Eve96]* Evers M., Chang P.Y., Patt Y.N**. - *Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches*, ISCA '96 (Ann. Int.'L Symp. Computer Architecture)

***[Gal93]* Gallant S.I**. - *Neural networks learning and expert systems*, The MIT Press, 1993

***[Koh95]* Kohonen T., et al.** - *Learning Vector Quantization (LVQ). Program Package Ver. 3.1*, Helsinki University of Technology, SF-02150 Espoo, Finland, 1995

***[Mud96]* Mudge T.N., et al.** - *Limits of Branch prediction*, Technical Report, Electrical Engineering and Computer Science Department, The University of Michigan, Ann Arbor, Michigan, USA, 1996

***[Pan92]* Pan S.T., So K., Rahmeh J.T.** - *Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation*, ASPLOS V Conference, Boston, October, 1992

***[Per93]* Perleberg C., Smith A. J**. - *Branch Target Buffer Design and Optimisation*, IEEE Transactions on Computers, No. 4, 1993.

***[Ste96]* Steven G. B. et al.** - *A Superscalar Architecture to Exploit Instruction Level Parallelism*, Proceedings of the Euromicro Conference, 2-5 September, Prague, 1996.

***[Vin99]* Vintan L., Armat C., Steven G.** - *The Impact of Cache Organisation on the Instruction Issue Rate of a Superscalar Processor*, Proceedings of Euromicro 7th Workshop on Paralle l and Distributed Systems (http://www.elet.polimi.it/pdp99/), Funchal, Portugal, 3rd - 5th February, 1999

***[Yeh92]* Yeh T., Patt Y.** - *Alternative Implementations of Two Level Adaptive Branch Prediction*, 19 th Ann. Int.'L Symp. Computer Architecture, 1992