Neural Branch Prediction

by Arun Lakshminarayanan and Sowmya Shriraghavan

19th April, 2004.

Introduction:

Branch misprediction latency is the most important component of performance degradation as microarchitectures become more deeply pipelined. Branch predictors must improve to avoid the increasing penalties of mispredictions.

Branch predictors on neural learning are the most accurate predictors in the literature, but they are impractical because the advantage of the extra accuracy is nullified by high access latency. This latency is due primarily to the complex computation that must be carried out to determine the excitation of an artificial neuron.

In this paper, we study a new neural branch predictor proposed by Daniel. A. Jimenez, published in the 36th International Symposium on Microarchitectures. This new predictor solves the problem from both directions:

- It is more accurate and
- Much faster than any of the previous neural predictors.

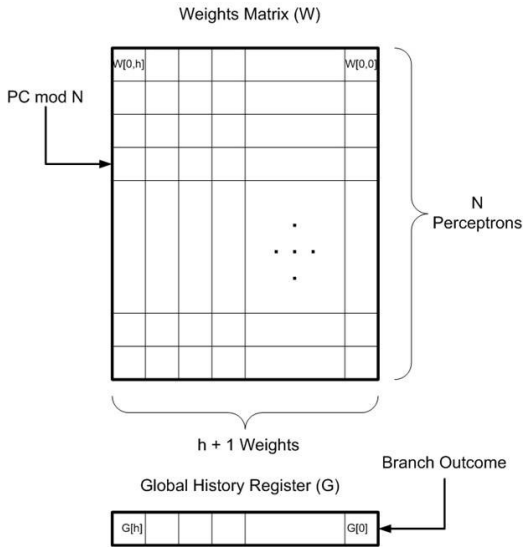
The new neural predictor improves accuracy by combining path and pattern history to overcome limitation inherent to previous predictors. It uses a different prediction algorithm that would allow parallel execution of instructions during every prediction, thereby keeping the latency low. In fact, the fast path-based neural predictor has a latency comparable to the predictors from industrial design and hence is a far superior predictor.

Neural Predictors:

Perceptron Predictor:

The Perceptron predictor uses perceptron learning to predict directions of conditional branches. It is a correlating predictor that makes a prediction for the current branch based on the history pattern observed for the previous branches.

A perceptron predictor uses a N*(h+1) matrix W of integer weights, where N and h are design parameters. Weights are typically 8-bit bytes. Each row of the matrix is an (h+1) length weights vector. Each weights vector stores the weights of one perceptron that is controlled by perceptron learning. In a weights vector w[0..h], the first weight w[0], is known as the bias weight. Thus, the first column W contains the bias weights of each weights vector. The boolean vector G[1..h] represents the global history shift register that contains {taken, not_taken} values, to store the outcome of the previous h branches.



To predict a branch with an instruction memory address M – we do a (M mod N) operation which would index onto a particular row in the W matrix. This row is an artificial perceptron that is responsible for prediction that particular branch. Note that a single perceptron could be responsible for the prediction of multiple branch instructions. The prediction is made based on the weight values in that row and on the outcome of the most recent "h" branches, which is stored in the global history register G.

Once the outcome is known, we train the perceptron that was responsible for the prediction in a specific way depending on whether the prediction was correct and also based on the pattern history stored, that is stored in the global history register G. This is accomplished by updating the weights in the corresponding row of the W matrix.

The training algorithm takes an integer parameter theta that controls the tradeoff between long-term accuracy and the ability to adapt to phase behavior. It has been empirically determined that choosing theta = [1.93*h + 14] gives the best accuracy, where h is the history length. Thus, for a given history length, theta is a constant.

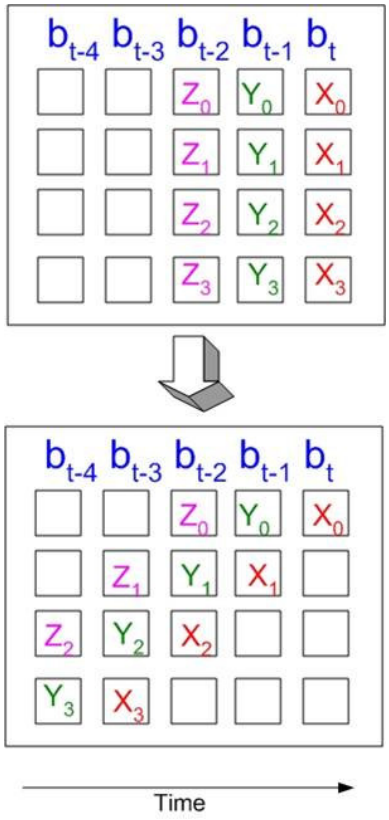
Fast Path-Based Neural Branch Predictor:

Our alternative to the perceptron predictor is a neural predictor that chooses its weights vector according to the path leading up to a branch, rather than according to the branch address alone. Like the perceptron predictor, it uses a weight matrix W of size N*(h+1) and a global history shift register G[1..h] to store the pattern history.

The technique has two advantages.

- Latency is mitigated because computation of y_{out} can begin in advance of the prediction, with each step proceeding as soon as a new element of the path is executed.
- Accuracy is improved because the predictor incorporates path information into the prediction algorithm.

Thus the prediction for a branch "b_t" at cycle "t" starts (t-h) cycles earlier. At every prediction step, the computation for the next "h" successive branches would be in progress. For a given branch, the prediction is split into "h" stages – each one involving an addition or a subtraction. Since all of the "h" different computations that need to be performed at a prediction step are independent, they can be executed in parallel, which is how we are cutting down on the latency for prediction. So essentially it is just the time involved to carry out one addition operation.



The picture at the top is for a perceptron predictor when all of the computation for a given branch is started and completed in one step. The picture at the bottom is for a neural predictor which staggers the computation into "h" different stages For example, let us consider the branch b_t. The computation for this branch involves adding the weights X_0, X_1, X_2 and X_3 is marked in red, with appropriate signs obtained, from the G vector. Note that the computation for this branch (b_t), starts at the same cycle when the branch b_{t-3} is being predicted.

Thus during every prediction process(b_{t-3}, b_{t-2}, b_{t-1} etc), h different computations are performed – which are all independent and hence can be executed in parallel, there by tremendously cutting down on the latency for prediction.

A detailed description of the prediction and update algorithms is provided in the research paper by Jimenez.

Experiments and Results

We implemented the perceptron and the path-based neural predictors in the simple-scalar simulator. We designed various experiments that would test different aspects of these predictors.

We decided to simulate non-speculative branch prediction, because the "prediction" and the "update" algorithm become far too complicated for a generalized case that would allow speculation. In fact, the research paper only mentions that these predictors can work for the speculative case as well – but the intricacies of the implementation were not discussed.

We ran our tests on sim-bpred and sim-outorder to obtain performance results. We used "cc1" and the "go" benchmarks for testing.

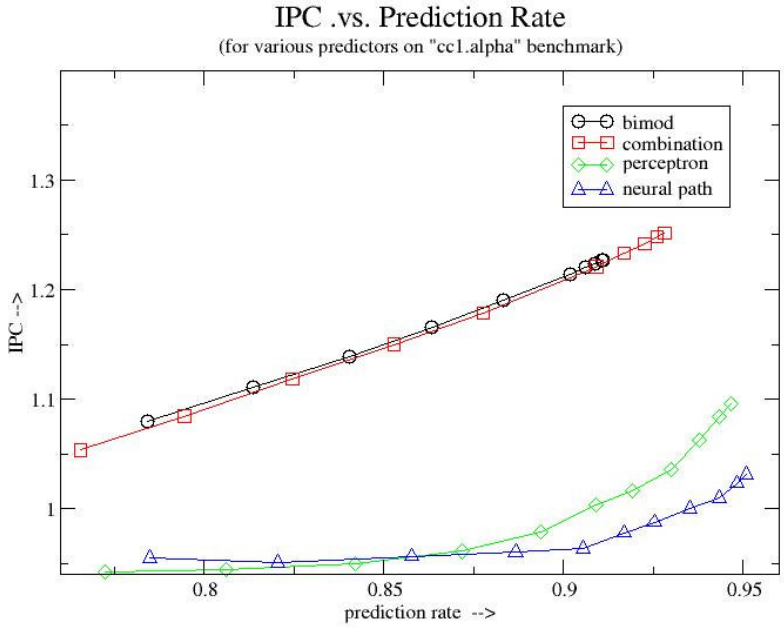
Since each of the test runs typically take about 4/5 minutes to complete, we decided to automate the testing procedure. For every experiment that we came up with, we had written python scripts that would automatically run different test cases one after the other, gather all the results and compile it into one output file – that we used to make tables and plots.

Experiment # 1 : Prediction Rate .vs. IPC

From a performance stand point, IPC is an excellent metric to measure performance. However, for the perceptron and the fast path-based neural predictor that we're talking about, the prediction latencies are extremely hard to simulate using the simple-scalar simulator. To do a latency analysis, we need to simulate the gates and switches at the hardware level using H-Spice or a similar software.

In this experiment we study how prediction rate is related to IPC.

Given below is a graph that shows how the IPC varies with prediction rate, for different branch predictors. For each predictor we increased the memory size – gradually from 64 bytes to 64KB (in multiples of 2) and measured the prediction rate and IPC values for each case, using the sim-bpred and sim-outorder programs respectively.



In general, we observe that as we increase the prediction rate, the IPC also increases. The relation is almost linear for the bimod and the combination predictors.

For the perceptron and the path-based neural predictor the IPC measured by the simple-scalar simulator is unrealistic. For instance, we have seen that for the path-based neural predictor, every prediction step involves a parallel computation (involving additions) for "h" successive branches. There is really no way that the simple-scalar simulator could take that into account, that fact. That is just the way the prediction algorithm is implemented at the hardware level. So in reality, the shape of the path based neural predictor would be different – and probably curving up much more rapidly than the bimod or the combination predictor.

One theory that would explain the shape of the curve for the perceptron and the neural-path predictor is that – the simple scalar simulator could be timing the prediction and update algorithms while computing the IPC values, using system timers and interrupts.

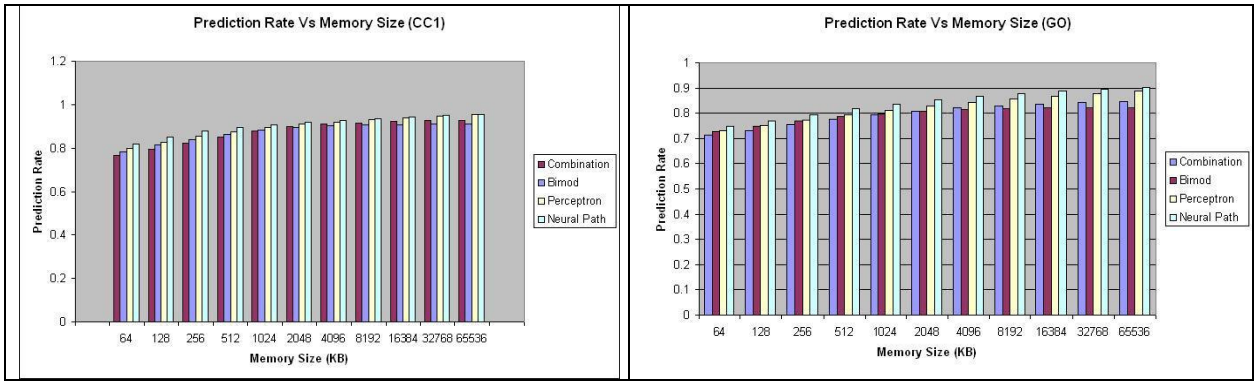
Obviously since much more computation in involved for the prediction and the update algorithm in both cases, the gain in accuracy is nullified by the increase in the computation time – and so the overall IPC does not increase so much and almost remains flat for most part. This is in fact true for the perceptron predictor.

However for the path based neural predictor, the update algorithm is executed in parallel – and hence will not involve an increase in computation time. So one can expect the neural path predictor to curve upward much more rapidly than both the bimod and the combination predictors.

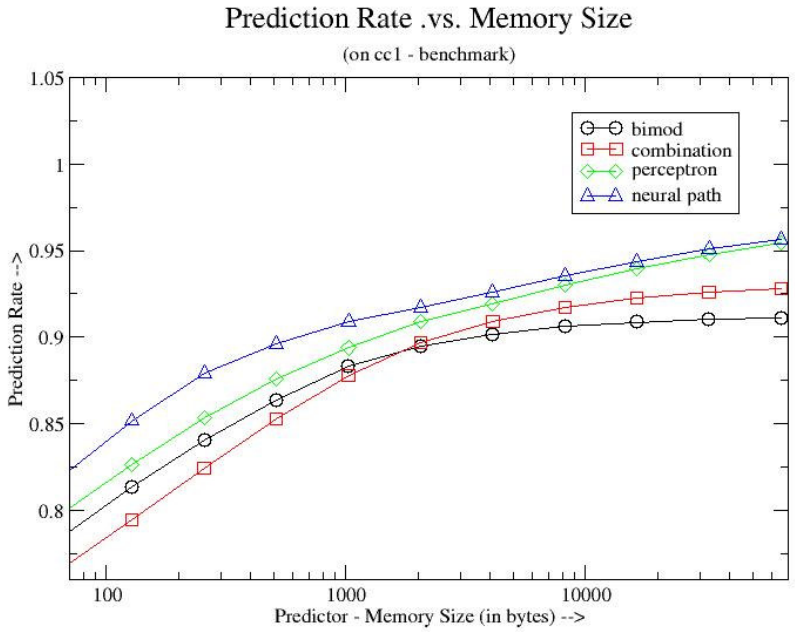
Experiment #2 : Comparing the performance of the neural-path predictor against other predictors used in industrial designs:

In this experiment we study the performance (in terms of the prediction rate) of the path based neural predictor and compare it against the performance of the bimod and combination predictors (used in industry) and against the perceptron predictor which is also a neural predictor and is known to have a high prediction accuracy.

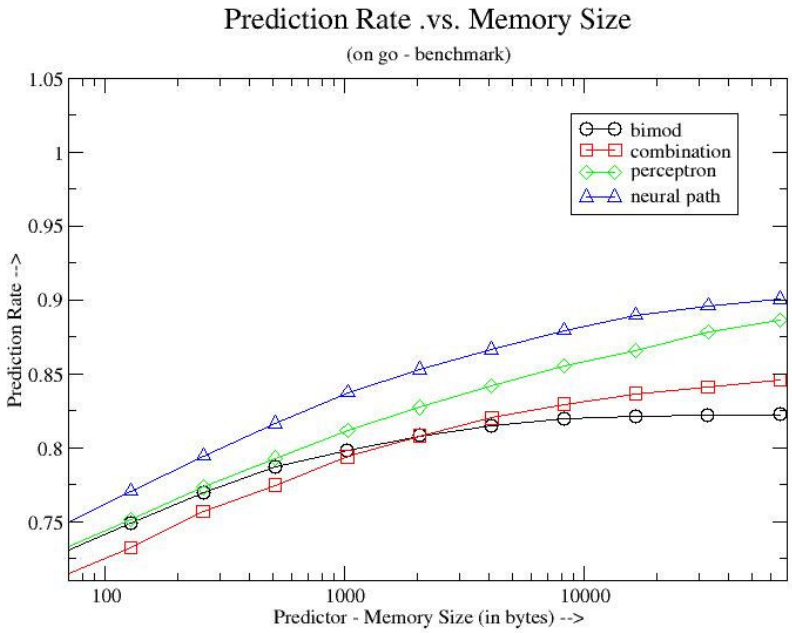
Results:



Performance on "cc1" benchmark:



Performance on "go" benchmark:

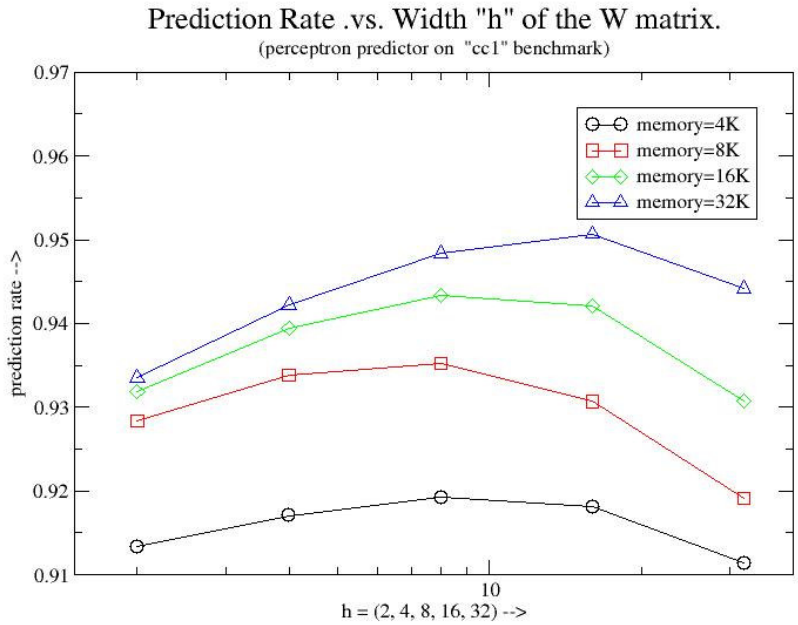


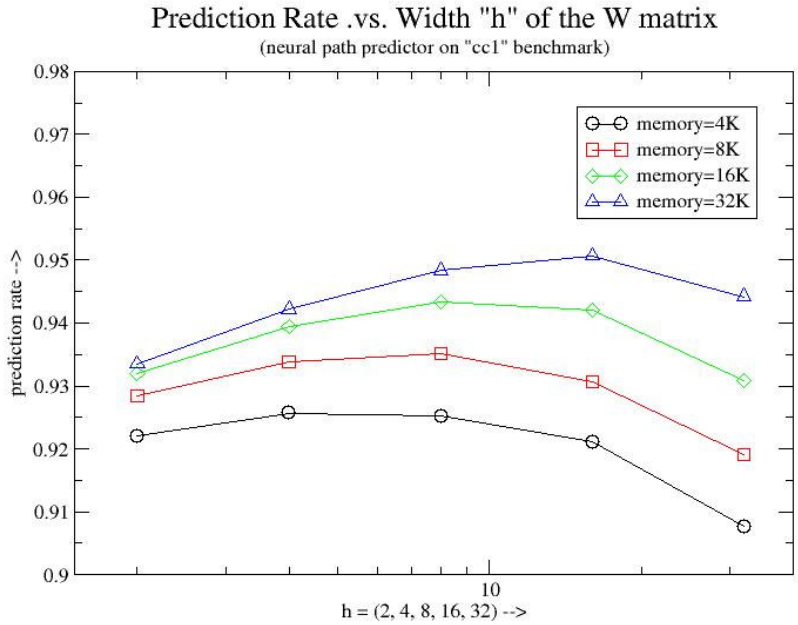
Thus, we observe that for a given memory size, the path-based neural predictor beats the performance of both the bimod and the combination predictor, and even the perceptron predictor.

The fact that it is a neural predictor – makes it a lot more powerful compared to any of the predictors for industrial design. On top of that, it bases its prediction on both the path-history and the pattern history which makes it far more superior compared to any of the other previously known neural predictors such as the perceptron predictor.

Experiment #3: Achieving maximum prediction accuracy for a given memory size, for the perceptron and the path-based neural predictor:

The weight matrix W has "N" rows and "h" columns. We increase the correlating history length "h" gradually, while keeping the product "N*h" a constant, so that the predictor would use the same amount of memory in each case. We study how the prediction rate changes as we change the pattern history length.





For both the perceptron and the path-based neural predictor, we observe that as we increase the history length “h”, the prediction rate increases, initially – reaches a maximum value and then decreases.

This is because, by increasing “h” we are able to capture more of the pattern history and correlate better. However as we increase “h”, we are losing out on the number of rows in the W matrix. (since we keep N*h a constant).

We have already seen that each row in the W matrix corresponds to an artificial perceptron or a neuron, depending on which predictor we are talking about. Thus, losing out on the number of rows means having fewer perceptrons or neurons to make predictions. So, now we would have more branches that would map on to the same neuron or the perceptron. Beyond a certain point, this would lead to too much interference and it becomes destructive, hurting the overall prediction accuracy.

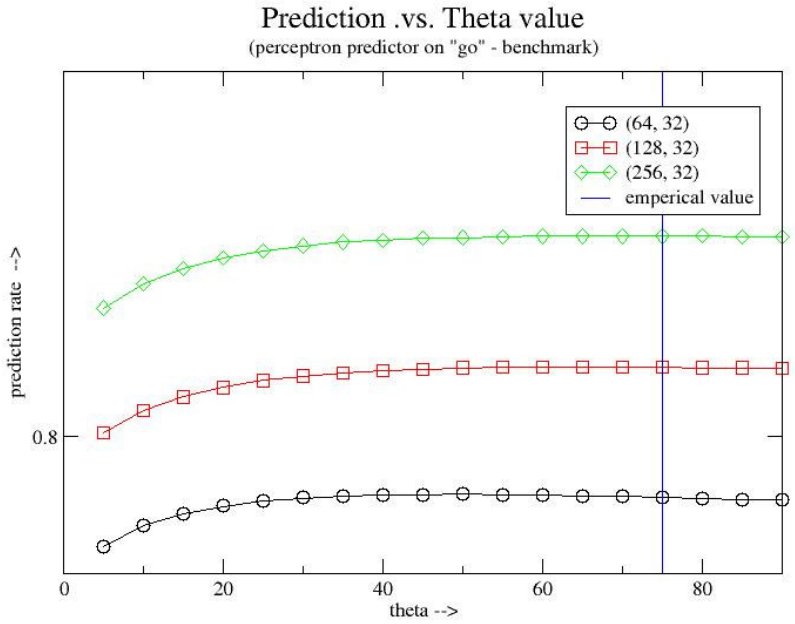
Thus, as we increase “h”, there is a clear trade off between better correlation associated with the ability to capture more of the pattern history, and the destructive interference caused because there are fewer perceptrons or neurons now than before, to predict the same number of branches.

Experiment #4 : Determining the optimal value for theta:

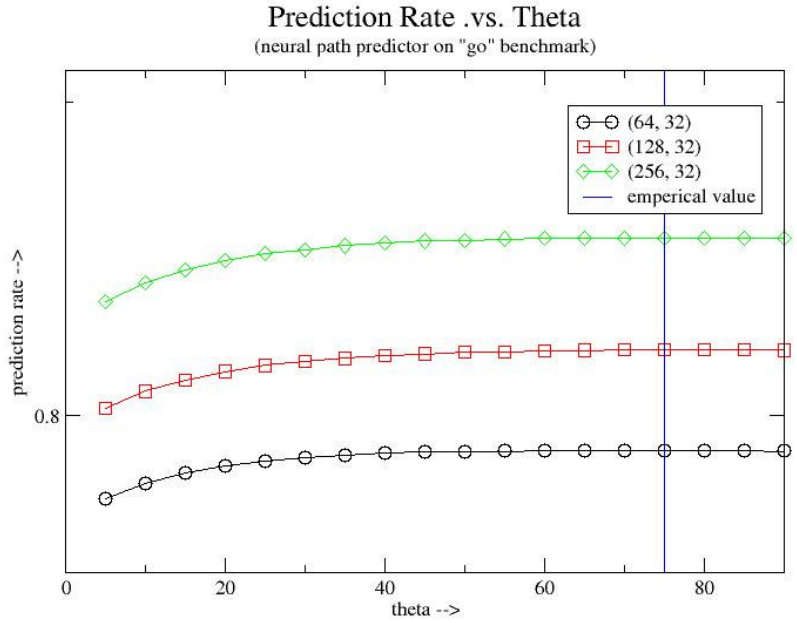
The training algorithm for both the perceptron and the path-based neural predictor takes an integer parameter theta that controls the trade off between long-term accuracy and the ability to adapt to phase behavior. Based on empirical analysis, it has been suggested that $\Theta = [1.93h + 14]$, gives the best accuracy.

In this experiment, we vary θ value for the predictor and observe the prediction rate. Keeping “h” fixed, we repeat the procedure for different “N” values to obtain the following results.

Perceptron Predictor:



Path-based Neural Predictor:



The different curves in each graph are for different (N,h) values. The history length h is kept a constant, while the number of rows was changed.

For both perceptron and the path-based neural predictor, up to a certain value, the prediction rate increases as we increase theta and flattens out after that. As we keep increasing the theta value still further, the prediction rate begins to decrease. For h=32, this critical value was observed to be about 75 which is the exact same as that predicted by the empirical formula.

The larger the theta value, the larger would be the absolute values of the weights in the W matrix. So suddenly if the program suddenly enters a new transitional phase in which the branches behave differently (follow a different pattern), it would take a very long time for the predictor to adapt to the new phase. However in the long-run, if the branches do not enter any transitional phase in which they tend to behave differently, a large theta value would not hurt at all and in fact is likely to help.

So placing an upper bound on theta value helps the predictor to be responsive enough to any sudden change in phase, while compromising a little bit on the long term accuracy.

Conclusion:

Based on the results that we obtained from different experiments, we can conclude that the path-based neural predictor is a much more powerful predictor, compared to any of the known industrial predictors or other previously known neural predictors such as the perceptron predictor.

The high prediction rate of the path-based neural predictor can be attributed to the fact that it makes its prediction based on both pattern-history and the path-history.

Since the computation that needs to performed for predicting a given branch is staggered and performed in multiple stages ("h" stages – where h is the length of the global history), the computation is started a few cycles earlier. At every branch prediction cycle, the computation for "h" successive branch predictions would be in progress, all of which are independent and hence can be performed in parallel.

This is exactly how, we are able to cut down on the prediction latency of this neural predictor.

Given its high prediction accuracy and latency comparable to those predictors in use, in the industry – it is likely to be the next generation of predictor to be used in the industry.

References:

[1]. Daniel A. Jimenez. Fast Path-Based Neural Branch Prediction, *36th International Symposium on Microarchitecture*, 2003.
[2]. H.D. Block. The Perceptron: A model for brain functioning. Review of Modern Physics.
[3]. Marius Evers et al. An analysis of correlation and predictability: what makes two level branch predictors work, *25th Annual International Symposium on Computer Architecture*, July 1998.
[4]. Ravi Nair. Dynamic path-based branch correlation, *28th Annual International Symposium in Microarchitecture*, 1995.
[5]. Lucian N Vintan and M. Iridon. Towards a high performance neural branch predictor. Proceedings of the *International Joint Conference on Neural Networks*, 1999.