ASSIGNMENT-3: BRANCH PREDICTION USING ML

Sai Gautham Ravipati EE19B053[1], **Shashank Nag EE19B118**[1]
[1]Department of Electrical Engineering, Indian Institute of Technology, Madras

1 Introduction

Branch instructions constitute an important part of the instructions pool, because of their effects on the performance of pipelined and super-scalar processors. Branch predictors are used to improve the pipeline flow and eliminate obvious stalls due to branch instructions. But on a branch mis-prediction, the penalty imposed is severe due to the fact that state of execution has to be rolled back. Furthermore, the mis-prediction penalty increases with increase in the pipeline depth, as the prediction happens at the instruction fetch stage, while the stage after which the speculation is resolved is moved farther. Hence, there have been important developments on accurate branch predictors, which mainly operate on the previous branch outcomes to predict the current branch decision. There exists techniques which update a counter to maintain the state, to those which incorporate the local and global branch histories to provide accurate branch predictions. Through this assignment, current predictors like bimodal, gshare, perceptron and hashed perceptron have been reviewed. Taking into consideration, the precedence set by machine learning algorithms on binary classification, analysing stream data etc., different machine learning approaches like Parallel Recurrent Networks, Convolution Neural Networks etc., have been analysed to solve the problem of branch prediction. Future implications of these techniques has been discussed in detail. All the simulations were carried out using the ChampSim simulation environment.

2 Existing Policies

The existing policies for branch prediction could be broadly classified as those maintaining the state using counters (gshare, bimodal), while the other set involves considering it as a classification problem, using weight based accumulation as in the case of perceptron and hashed perceptron. While the regular dynamic schemes require exponential scaling of hardware to capture the history length, the same could be performed using linear scaling of hardware, when neural network based dynamic predictors are utilised. Each of the policies has been illustrated upon briefly. The profiling results of these policies has been presented in Figure-1.

2.1 Bimodal

In the case of a bimodal branch predictor, a 2-bit counter is maintained to update the state of branch being taken/non-taken with 11/00 corresponding to high-strength regions of branch being taken/not-taken, and the other two states are used to switch between the cases of mis-predictions. The PC is hashed to index into the table maintaining a set of counters, which are updated based on the branch result and the current state of the counter. This methodology takes longer time to learn and switch to other predictions because of adjacent states giving out the same prediction. Furthermore, the bimodal branch predictor gives weight to the history of current branch, rather the history of some other branch. However, in the real world scenario, different branches could be correlated and hence, this motivates the need for correlated branch predictors like gshare. The same can be verified from the MPKI of both the predictors where gshare has a better MPKI in all the cases in comparison to bimodal predictor.

2.2 Gshare

In this architecture, the history of last m-branches is taken into consideration by maintaining an m-bit global history register, to account for recent behaviour of other branches. This global history is XOR-ed with the pc to make sure that different combinations of (XOR/pc) go into a different entry of the table. However, this methodology suffers if the global history is polluted by uncorrelated branches. For better prediction accuracy a larger history buffer might be required, which shall increase the resources required exponentially. The perceptron based predictors try to solve this problem, by making use of long branch histories, and since the resources required scales linearly, this is permitted[1]. It can be observed that the MPKI values of perceptron based predictor are better than the Gshare predictor because of this reason.

2.3 Perceptron based methods

In a perceptron based predictor, an accumulated sum of the current global history is computed with the weights corresponding to the process history, in-order to make a decision. Based on the branch result the weights are updated positively if the corresponding entry in the global history correlates with the branch outcome. By this means a set of weights are learnt at run-time and are used to predict the results of upcoming branches. The average MPKI of perceptron based prediction is 8.968 which is far better than the other dynamic branch prediction methods. The hashed perceptron predictor uses the idea of multiple independently indexed tables of perceptron weights, as well as used geometric history lengths to obtain better prediction accuracy. The average MPKI of hashed perceptron based predictor is 8.573, which is better in comparision to the other three methods discussed previously, where perceptron, gshare, bimodal have an average MPKI of $8.968, 9.254, 12.312$ respectively.

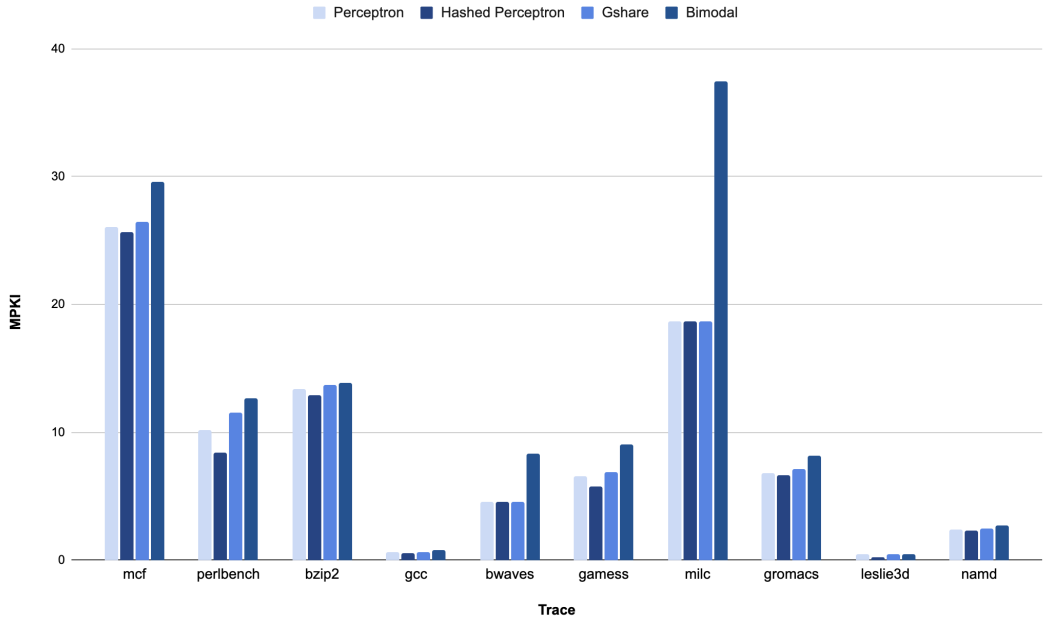

Figure 1: MPKI data for different traces

3 Custom Policies

In the following few sections, several custom policies that have been experimented are presented. One important case where the perceptron based policies might not work is the issue of linear separability. In the case of perceptrons, the patterns they can learn is limited by their nature of separating the plane linearly - hence, they can capture dependencies like AND, but not something like XOR. Although they do well in practise because of limited occurrences of these cases, they still fail to detect a few hard to detect branches. To mitigate the same, approaches like multi-layered perceptron, RNN (a variation where implementation can be parallised for latency is considered), and a simple single-filter CNN have been evaluated.

3.1 RNN based branch prediction

Recurrent Neural Networks are good at capturing the patterns in a stream of data. The global history can be considered as a stream, and a recurrent neural network could be used to operate on the same to infer the output prediction. However, they are highly sequential, and in the case of a global history of length 32 being considered, this further increases the latency to produce the prediction outcome. Hence, a variant of RNN, the Sliced RNN has been adopted[2]. In this case, the input global history is sliced to sub-slices of length 8, where 4 such stages could operate in parallel. The operating cell of a particular sub-stage with slice size of 8, has been presented in Figure-2. For a prediction on global history of length 32, outputs from 4 of such blocks are operated by an RNN cell to give the final prediction.

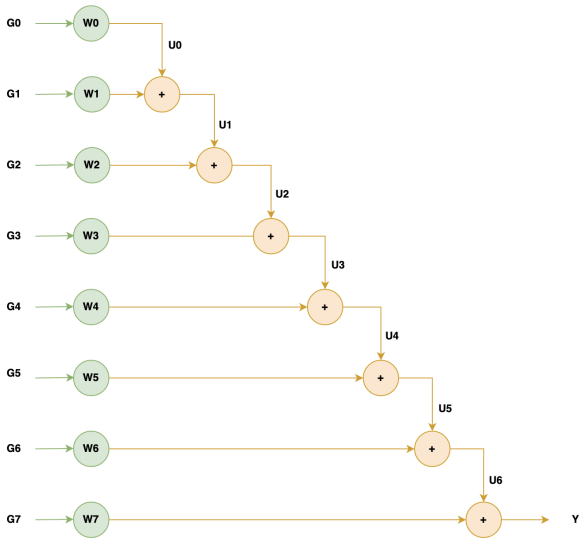

Figure 2: Sub-cell being used in a Sliced Recurrent Neural Network, with slice size of 8

The PC is used to hash into a weight table to obtain the values $w0 - w7$ and $u0 - u6$, and these are used to compute the output prediction. On update, those weights which contribute positively to the actual result are incremented while the other are penalised. Through this approach the network is being trained online, to reduce the number of mis-predictions. The advantages of this

approach include faster training over the cases of multi-layered perceptron or CNN as there is direct flow of weights from the input. Furthermore, the RNN structure, tries to hold the information over larger histories and could separate non-linear spaces when proper activation functions are used. The average MPKI obtained for profiling on the base cases was 9.2064. This is although higher than the case of perceptron, this was obtained when a linear activation function was used. Further, the ML based networks suffer from inaccuracy during the learning period, which is solved in most cases by a standalone predictor during this phase. In this case, the previously mentioned MPKI was obtained without any support for non-lineraity or during the learning phase, and this performs better than the regular dynamic branch predictors. Further possible optimisations could be properly tuned hyper-parameters, as well as support during learning phase. Given the fact that multiple stages could be parallelised, when implemented on an actual hardware, this could be exploited to reduce the access latencies. However, in this methodology, since we are using fixed length branch history, this might be difficult to identify some of the hard-to-find branches - approaches like LSTMs and GRUs have been proposed to tackle the same.

3.2 2-layer Perceptron

Taking into consideration the requirement of variable length of history being considered, as well as from the architecture of hashed perceptron, the following architecture has been proposed based on a set of simple perceptrons. The data flow has been presented in Figure-3a. It could be noted that geometric lengths of inputs are being considered for each of the perceptrons nodes, which shall provide a better representation of global branch history to identify hard-to-predict branches. The output is computed using a smaller perceptron operating on the activations produced by the previous set of perceptrons. During the update stage, the output of the second set of neurons, is matched with the intended branch result, and on a match the weights of the second stage are updated positively. For the weights of first stage update decision is made based on the input global history. The average MPKI obtained for this case is 8.9516 which is slightly better than that of perceptron, owing to the geometric lengths of history vectors being considered. However, the activation functions being used are still linear and can be non-linearised with appropriate fine-tuning to gain improved performance. Furthermore, because of the linear structure of perceptron, it makes it difficult to capture complex structures in the branch history. To reduce the access latencies, the individual nodes of a perceptron can be computed concurrently on the actual hardware implementation.

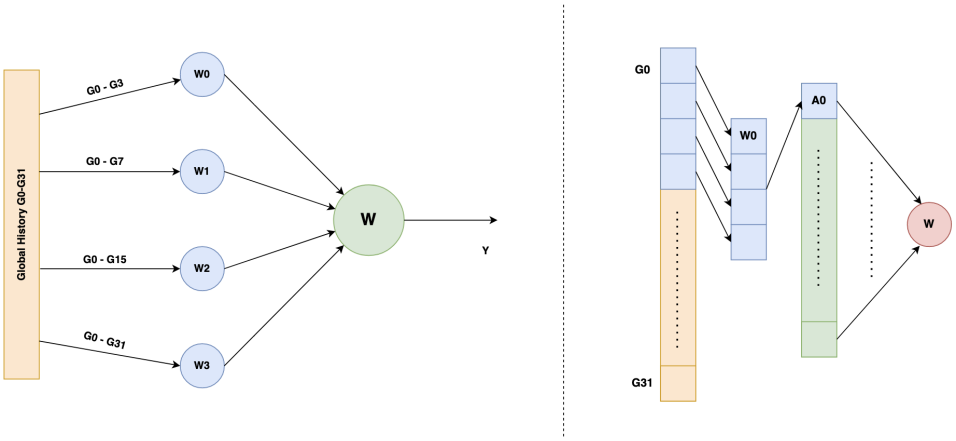

Figure 3: a) Data-flow of a 2-layer perceptron, b) Data-flow of a single filter CNN

3.3 CNN based Branch Predictor

In this approach, the input global history is convolved with a simple 4-weight filter, so that the entries can be filtered out for irregularities before passing to the perceptron. The weights for the particular branch are obtained from the process history table index by the hashed pc. After the computation of node-1 is complete, this requires the usage of a perceptron in stage-2 to compute the prediction. This turns out to be advantageous since, a filtered version of global history is presented to the perceptron, so that it can operate more precisely on cases like hard-to-predict branches, or the case of input dependent branches, where obtaining a correlation turns out to be difficult. Not only does this improve the quality of branch history, but also solves the problem of linear separability which existed in the case of perceptron based predictor. During the update phase, the weights of the perceptron are updated based on the activations obtained after convolution filtering. The filter weights are updated based on the input activations, so that the output prediction is made closer to the actual result on a mis-prediction. The average MPKI obtained for this case was 8.8986, which is better than the case if perceptron as well as the 2-layer perceptron. This indicated the usefulness of using a CNN based predictor which could be tuned further, in-order to achieve substantial gains. However, this particular approach requires sufficient hardware overhead, to compute the activations of intermediate layers parallely. It can be noted that, even after the addition of an additional filter, the prediction outcome is lower than that of hashed perceptron, and the CNN based approach, isn't able to detect some of the hard-to-predict branches. But given the larger scope of Neural Networks this could be analysed further, on tweaking them for performance improvement. Furthermore, in most of the cases, these machine learning approaches require prior offline training[3] for better operation, and performing the same might improve the results of prediction.

In addition to the above mentioned optimisations additional optimisations like using a dynamic threshold setting have been performed. Furthermore, the bits are quantised to represent the case of actual hardware. To index into the process history table, the appropriate branch direction has also been incorporated to account for the direction of jump. The results obtained for each of these cases have been presented in Figure-4, 5.

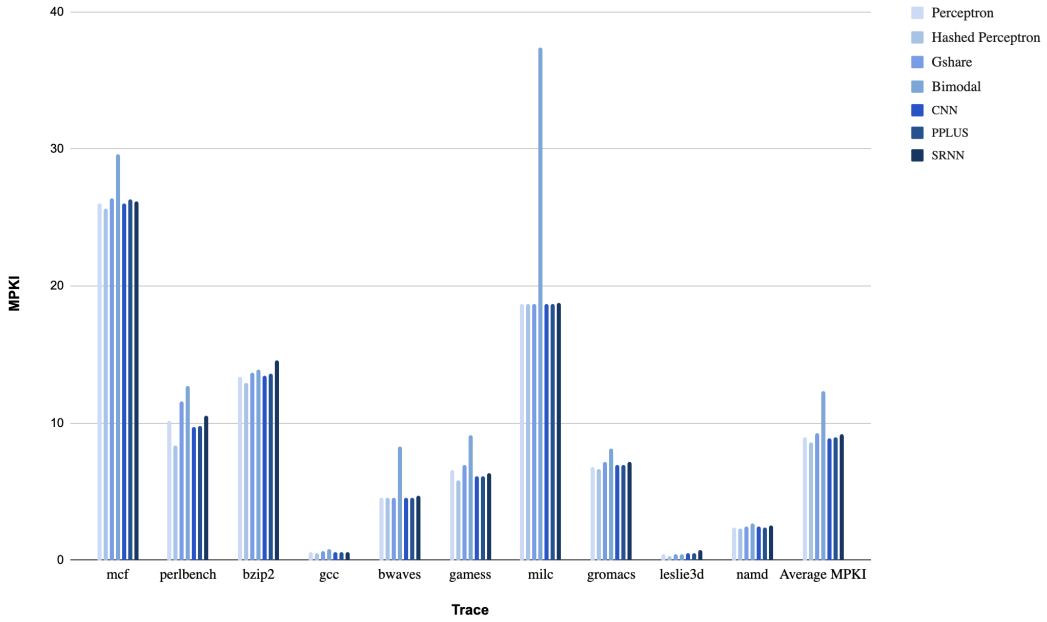

Figure 4: MPKI values for different input traces

Trace	Perceptron	Hashed Per.	Gshare	Bimodal	CNN	PPLUS	SRNN
mcf	26.0458	25.6371	26.4306	29.6266	25.9922	26.2853	26.1415
perlbench	10.1552	8.39628	11.5568	12.6708	9.6846	9.80872	10.5196
bzip2	13.3774	12.9423	13.6792	13.8728	13.4231	13.62	14.5746
gcc	0.61248	0.51766	0.6419	0.8116	0.5769	0.61756	0.56234
bwaves	4.52892	4.52892	4.52892	8.2932	4.52896	4.52892	4.6697
gamess	6.60316	5.80062	6.92208	9.08712	6.15684	6.11286	6.36452
milc	18.7089	18.7087	18.7088	37.4172	18.7088	18.7088	18.7718
gromacs	6.8245	6.64884	7.14558	8.1641	6.9698	6.97456	7.16658
leslie3d	0.4712	0.25394	0.4712	0.4712	0.51764	0.47874	0.76686
namd	2.35662	2.29946	2.45428	2.70974	2.4278	2.3806	2.52732
Average MPKI	8.968418	8.573382	9.253936	12.312436	8.898664	8.951606	9.206482

Figure 5: Numerical values of the profiled MPKI data

4 Conclusion

In this assignment, multiple branch prediction schemes have been analysed and profiled using the SPEC benchmarks. The following inferences were drawn:

- The hashed perceptron predictor performs the best amongst the existing predictors, followed by the perceptron. Amongst the newly implemented predictors the CNN predictor performs the best while the CNN and multi-layer perceptron have a lower MPKI in comparision to that of the perceptron predictor, over which they are improved on.
- The CNN based predictor doing the best can be attributed to its nature of filtering the global history and then operating the same, which shall provide stronger correlation amongst the branches, while removing the irrelevant uncorrelated branches from the scope.
- Furthermore, the structure of the CNN, allows to learn few hard-to-predict branches like conditionals dependent on an input which might be difficult to learn either in the case of SRNN or 2-layer network.
- Although the SRNN branch predictor doesn't perform up to the mark, it provides a possible direction of improvement because of its parallel nature in contrast to that of RNN. At the same time, incorporating advanced optimisations into it might improve the performance.

5 References

1 : Jimenez et al., "Dynamic Branch Prediction with Perceptrons"

2 : Zhang et al., "A Dynamic Branch Predictor Based on Parallel Structure of SRNN"

3 : Zangeneh et. al., "Branch Prediction with Multi-Layer Neural Networks"