2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT) | 978-1-6654-1005-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICICICT54557.2022.9917915

Exploration of Performance of Dynamic Branch Predictors used in Mitigating Cost of Branching

1st Akash Ambashankar
Department of CSE
KCG College of Technology
Chennai, India
akashambashankar@gmail.com

2nd Ganesh Chandrasekar
Department of CSE
KCG College of Technology
Chennai, India
cb.ganesh666@gmail.com

3trd Charan AR
Department of CSE
KCG College of Technology
Chennai, India
chintu0033@gmail.com

4th Sankar S
Department of CSE
KCG College of Technology
Chennai, India
sankar@kcgcollege.com

***Abstract*—Today, Branch Predictors have become a sophisticated component in all modern processors. While contemporary predictors have good accuracy, the cost of a misprediction is quite high. Hence, active research is being conducted towards improving the performance of branch predictors, while reducing the resources consumed. Modern computer architectures increasingly rely on speculation to boost instruction-level parallelism (ILP). As the pipelines deepen and instructions issued per cycle increase, the accurate branch prediction mechanisms become more and more important in increasing the processor performance. This study explores some branch predictors and compares them on the basis of their performance on certain benchmarks applications.**

***Index Terms*—Branch Prediction, Machine Learning, Neural Networks, Perceptrons**

I. Introduction

Branch prediction is a technique that is used to improve the performance of microprocessors. There are two types of branch prediction, namely Static and Dynamic Branch Prediction. Static predictors make raw predictions, without using any knowledge of previous decisions. On the other hand, dynamic predictors make predictions based on the outcome of similar decisions in the past. The logic for branch prediction is generally implemented using a dedicated hardware module, known as Branch Predictor Unit (BPU).

A two-way branch is usually implemented using conditional jump instructions. A branch may either be "not taken", wherein the program continues on the current path of execution, or "taken" where the execution jumps to the location of the second branch in program memory, It is not known whether a certain branch will be taken until the conditional jump has been executed in the pipeline. Fig. 1 illustrates the working of a branch predictor. Instead of encountering a conditional jump and then stalling until the conditional jump has been executed, a branch predictor tries to guess if the conditional jump is likely to be taken or not taken. Based on this guess, the branch to be taken is speculatively executed in parallel until the conditional jump enters the pipeline. During the execution of the conditional jump, it can be determined if the prediction

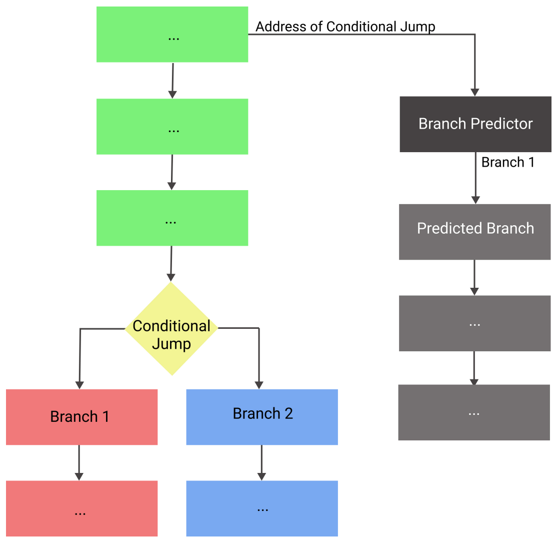

Fig. 1: **Branch Prediction Overview**

was correct. In the event that the guessed branch is incorrect, the speculatively executed instructions are discarded or flushed out, and the pipeline has to be restarted from the correct branch. This induces a bubble in the pipeline and a delay in execution. Fig. 2 indicates the pipeline during a correct prediction and incorrect prediction. The time that is wasted in case of a branch misprediction is equal to the number of stages in the pipeline from the fetch stage to the execute stage.

Modern CPUs are made up of deep-pipelined microprocessors, which increase instruction throughput, enable higher clock rates, improve parallelism, and thereby benefit the overall performance of the CPU. But, a consequence of deeply-pipelined microprocessors is the cost incurred when the branch predictor makes a misprediction. If a predicted branch is incorrect, the entire pipeline needs to be flushed, and in the case of deep pipelines, this can be an expensive process in terms of time as well as computational power. Hence there is

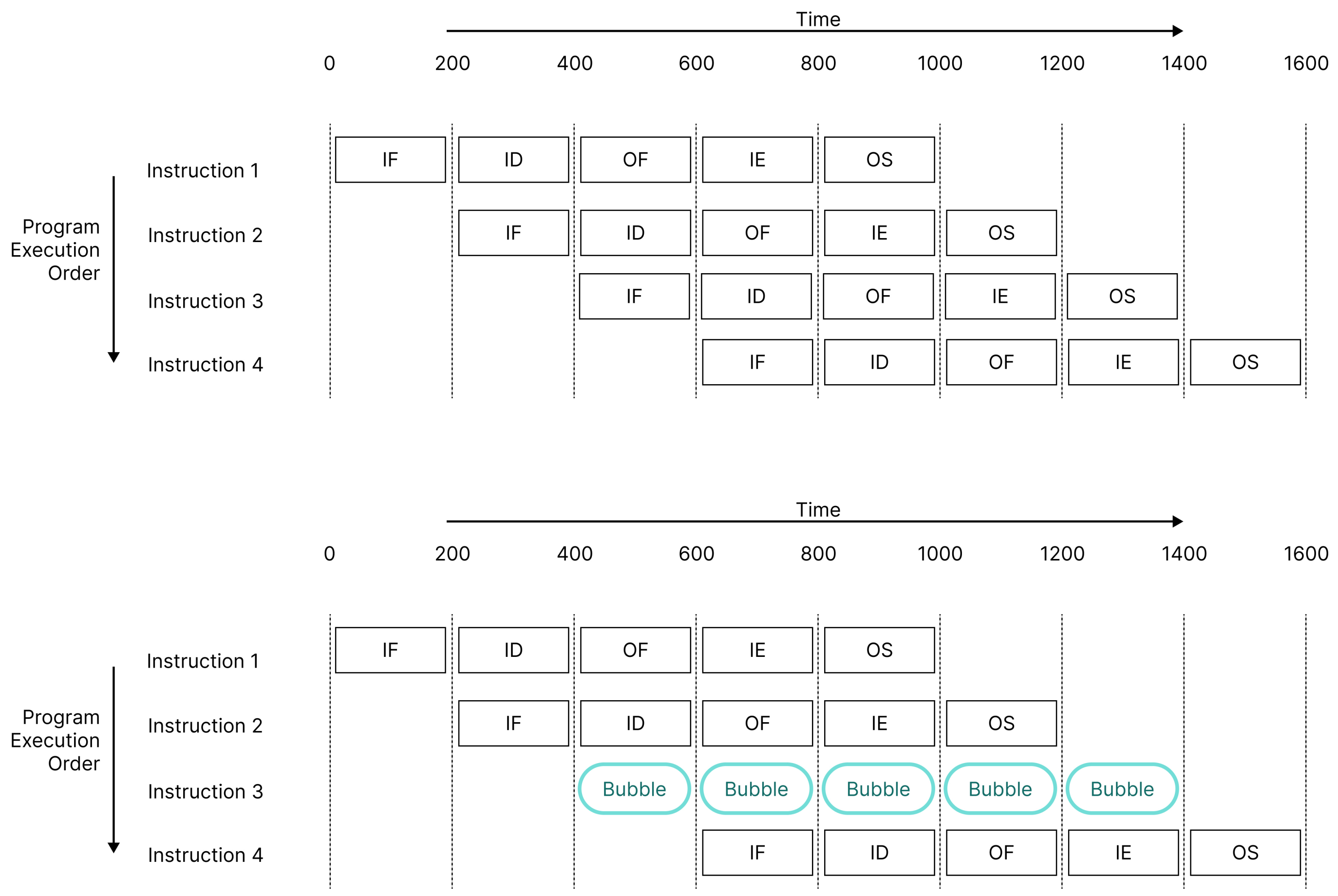

Fig. 2: **Instruction Pipeline: normal execution vs misprediction**

a need to keep improving the accuracy of branch predictors to avoid the penalties caused by misprediction.

In recent times, there have been many contributions toward improving the accuracy of branch prediction through neural methods, the most prominent method being the perceptron. This is a result of the perceptron's inherently superior predictive capabilities in combination with their training speeds which surpass those of neural networks. Within perceptron-based branch predictors, there are a plethora of variations such as the piecewise-linear branch predictor and bias-free branch predictor.

This comparative study implements five branch prediction strategies by testing them on various application benchmarks and evaluating them on the basis of metrics such as Accuracy and IPC. The rest of this paper is organized as follows: Section II provides an overview into the branch prediction strategies implemented in this study. Section III provides into the simulation environment as well as the benchmarks and metrics used in the results. Section IV presents the results along with discussion on the results. Finally Section V concludes the report.

II. Models

A. Bimodal Predictor

The Bimodal branch predictor is a well-known branch prediction model prevalent in the days before Machine Learning was introduced into branch prediction. The predictor value is based on the directions the branch traversed in the last few executed pipelines. Most recent studies show that the accuracy of the bimodal predictor increased exponentially by increasing the size of the branch history. The functionality of this predictor is far from random and most branches are either taken or not taken. So, the Bimodal branch predictor takes a side on this distribution of the branch behavior and attempts to predict taken from not taken branches. The default size for the bimodal branch predictor is 2 bit counter and these counters are indexed in the low order address bits program counter section.

For each taken branch the counter is incremented and for each not taken branch is decremented likewise. Bimodal has this feature where the counter value is neither incremented past three nor decremented below zero. By using this feature the bimodal predictor can either be taken or not taken based on the repeated branch values.

B. 2-level Adaptive Predictor

One of the most serious problems in branch prediction is when a branch mispredictions happen and the entire pipeline needs to be restarted after flushing the existing one. This tends to grow exponentially when the more superscalar processor is used for the pipeline. Numerous studies show which aspect of the different predictors functions better but don't mention the characteristics that makes the predictor function better.

When branch predictor uses correlations to produce a more accurate prediction score. But recent studies have shown that the Two-level predictor is not fully utilizing the correlations factor in its functionality.

If the correlations are taken for two to three branches it produces better accuracy than a two-level predictor. So if a two-level predictor takes advantage of single strong correlations for each of the branches in the pipeline, numerous studies conducted on this characteristic of the predictor show significant improvement in the accuracy score by 3.7% overall prediction and 1.7% on average for each prediction. In conclusion, when more number correlations for each branch are used in a larger set of branches yields better overall and average accuracy.

C. TAGE Predictor

Studies on increasing the accuracy of the conditional branch predictor show that combining predictors is the way forward. Most of these combination methods rely on several predictors components indexed with varying history lengths. The TAGE predictor uses tagged (partially) components, it relies on hit-miss detection as the computation function of the predictor. Also provides one of the best conditional branch prediction accuracies, in the area of equivalent storage budgets TAGE outperforms other predictors in the combined pool of predictors.

One of the most significant problems is the prediction of the indirect branches' accuracy. Here the TAGE principles can be directly applied to the indirect branches ITTAGE (Indirect Target TAgged GEometric history length) and this method outperforms the exiting direct target branch predictors. The TAGE and ITTAGE predictors feature a unique length of distinct history indices forming a geometric series. Thus combining them to create a single cost-effective predictor, sharing logic and tables. Resulting in the COTTAGE (COnditional and indirect Target TAgged predictor GEometric history length) predictor.

D. Perceptron Predictor

The perceptron is a method for branch prediction, which revolutionised the industry by replacing the two-bit counters previously used and providing increased accuracy. The source of this improvement is the access to a longer branch history as the hardware resources required by the perceptron scale linearly, rather than exponentially, with the history length.

Neural networks are known for their superior accuracy. Hence arose the question of whether the accuracy of branch predictors could be improved by replacing the standard saturating counters with neural networks. The challenge with neural networks is that they are extremely expensive and cannot be used in branch prediction. Hence, the authors experiment with using the perceptron, which is essentially equivalent to one unit or neuron of a neural network. Perceptrons train faster and are computationally less expensive as compared to neural networks, all while providing a similar accuracy.

This work proposes a two-level scheme that uses perceptrons instead of two-bit counters. Perceptrons can exploit long history lengths because the size of perceptrons scales linearly with the size of their inputs, which in our case is the branch history. Traditional two-level adaptive schemes use pattern history tables (PHTs), which grow exponentially with the history length. For example, in a 4KB hardware budget, traditional methods would be able to maintain a history length of 14 branches, whereas perceptrons are capable of maintaining up to 34 branches. The access to longer and richer branch histories leads to perceptrons making higher accuracy predictions.

The perceptron is defined as a learning device that takes a set of input values and combines them with a set of weights to produce an output value. These weights are learned through training. In the perceptron-based branch predictor, each weight is a function of the correlation between a past branch and the branch being predicted. Positive weights represent positive correlation, and negative weights represent the negative correlation. While making a prediction, each weight affects the prediction in proportion to its magnitude. If a corresponding branch is taken, weight is incremented, if it is not taken, weight is decremented. If the resulting sum is positive the output is taken and if the sum is negative the output is not taken. To this end, taken and not taken are represented by 1 and -1 respectively.

E. Bias-Free Predictor

Branch predictors based on neural networks from longer branch histories. Therefore, these predictors are highly effective only under large hardware budgets and fail to deliver the expected performance under 32-64KB storage. The ISL-TAGE is an example that shows longer branch histories (in the order of 2000 branches) result in better predictions. However, scaling a perceptron-based predictor from 64KB to 1MB storage causes longer computational latency and high energy consumption, which is not ideal for commercial use.

Gope et al. [6] propose a model that is designed to learn from and capitalize on correlations only between branches that are not biased. This model is known as the Bias-free branch predictor. Not all branches in the branch history provide useful context. Some branches, known as biased branches, resolve to the same result virtually every time. Thus, including them in the branch history doesn't directly contribute to the predictor's accuracy, rather these branches take up space that could be occupied by more useful non-biased branches.

In addition, the Bias-Free predictor also uses a recency-stack-like management policy for the global history register. This in combination with the bias-free branches in global history results in the neural network having access to much older branches and a significantly richer context. It also achieves comparable accuracy to that of TAGE predictors, which are known to employ longer branch histories.

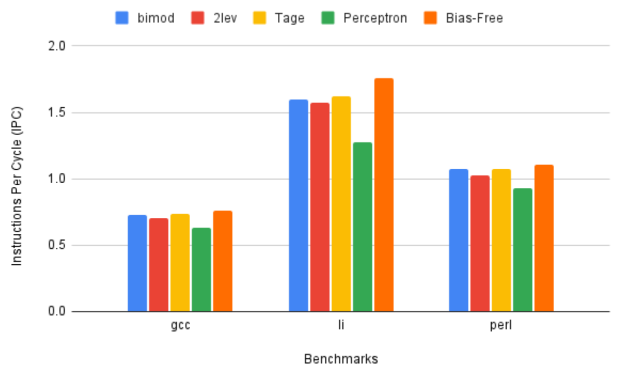

Fig. 3: **IPC of each predictor**

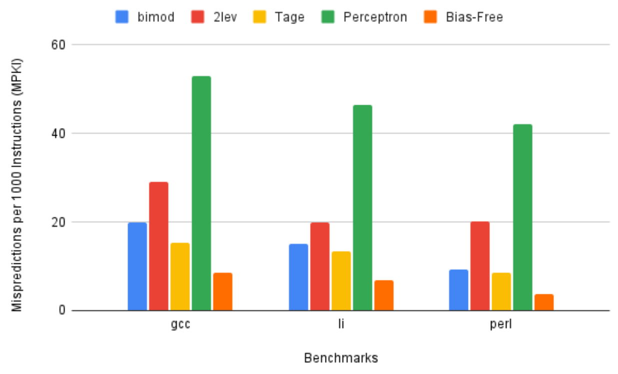

Fig. 4: **MPKI of each predictor**

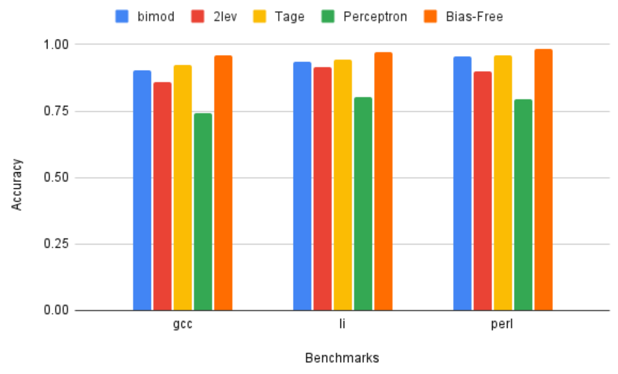

Fig. 5: **Accuracy of each predictor**

III. Methodology

The study conducts experiments using the SimpleScalar Simulator (version 3.0). Each predictor is implemented in the branch prediction unit of the simulator. The predictors are then tested on application benchmarks from Standard Performance Evaluation Corporation (SPEC) suite of benchmarks. The performance of each predictor is recorded and presented in Section IV.

A. Simulation Environment

All simulations are performed on the SimpleScalar Simulator, which is an architecture simulator developed by Todd Austin. SimpleScalar is designed to provide fast, flexible and extensible infrastructure for hardware modeling and software analysis. It reproduces the behaviour of a computer, thereby enabling development of system software such as branch predictors.

SimpleScalar offers a variety of configuration in its simulator. It can emulate the Portable Instruction Set Architecture (PISA), Alpha, ARM, and x86 instruction set architectures. There are also a number of simulator variants available such as *sim-bpred, sim-outorder, sim-profile, sim-cache,* each of which can be used as per the requirement. The Bimodal predictor and 2-level adaptive predictor are built into the simulator along with a few others.

The simulator used in this study is configured to the PISA architecture. The results are obtained using the *sim-outorder* simulator, which provides CPU information along with branch prediction stats.

B. Benchmarks

The Standard Performance Evaluation Corporation (SPEC) provides benchmark suites to evaluate performance and energy efficiency for computing systems. These benchmark suites are made for CPU, Graphics/Workstations, Servers, etc. This study makes use of three application benchmarks from the SPEC CPU benchmark suite, namely *gcc, li, and perl,* to evaluate and compare the various predictors.

C. Metrics

The performance of the branch predictors are evaluated using the following metrics.

1) Instructions Per Cycle (IPC): IPC is a measure of the average number of instructions executed in each clock cycle of the CPU. It is calculated by dividing the number of instructions by the number of clock cycles in the CPU. A higher IPC signifies better efficiency of the predictor.

$$IPC = \frac{Number\ of\ Instructions\ Committed}{Simulation\ Time\ in\ Cycles} \tag{1}$$

2) Mispredictions Per Kilo (=1000) Instructions (MPKI): MPKI is a measure of the average number of mispredictions for every thousand predictions. It is an indicator for the overall performance of the predictor on a particular benchmark. It is calculated by dividing the number of mispredictions by the total number of instructions and multiplying the result by 1000. A lower MPKI indicates a better performance of the predictor.

$$MPKI = \frac{Number\ of\ mispredictions}{Number\ of\ Instructions\ Committed} \times 1000 \tag{2}$$

TABLE I: Performance of each predictor over various benchmarks

Predictors	**Benchmark: gcc Metrics**			**Benchmark: li Metrics**			**Benchmark: perl Metrics**		
	IPC	**MPKI**	**Accuracy**	**IPC**	**MPKI**	**Accuracy**	**IPC**	**MPKI**	**Accuracy**
bimod	0.7254	19.9049	0.9025	1.5936	15.172	0.9359	1.0741	9.22	0.9547
2lev	0.7052	29.0939	0.8574	1.5703	19.848	0.9161	1.028	20.149	0.901
Tage	0.7371	15.275	0.9251	1.6249	13.42	0.9433	1.071	8.608	0.9577
Perceptron	0.6308	52.8978	0.7408	1.2716	46.505	0.8035	0.9317	42.078	0.7933
Bias-Free	**0.759**	**8.478**	**0.9585**	**1.7609**	**6.928**	**0.9707**	**1.1092**	**3.645**	**0.9821**

3) Accuracy: Accuracy is a measure of the rate of correct predictions of the branch predictor. It is calculated by dividing the number of correct predictions by the total number of predictions and multiplying the result by 100. A higher accuracy means the model's predictions are accurate to the actual branch outcomes.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Number\ of\ Branches\ Committed} \times 100 \tag{3}$$

IV. Results and Discussion

Each model was executed in the SimpleScalar simulator and the results for three benchmarks from the SPEC CPU benchmark suite were recorded. Table I shows the performance of each model across the various benchmarks.

On analysing the models' performance, it is evident that the Bias-Free predictor outperforms all other predictors across all metrics. Fig. 3 shows that all models have relatively low variation in IPC for *gcc* and *perl* benchmarks, as compared to *li*. This is an indicator that as total number of branches increases, the Bias-Free predictor performs better than other predictors. In addition, it also shows that the perceptron falls behind since the memory budget requirement increases with the increase in number of branches. The TAGE predictor has a better IPC than the other three models and falls just short of the Bias-Free predictor. This is a result of the longer global history of the TAGE predictor.

Fig. 4 shows that the Perceptron predictor suffers from a relatively large MPKI in all three benchmarks. The 2-level Adaptive Predictor and Bimodal Predictor have half the MPKI of the Perceptron. The TAGE and Bias-Free predictors have the lowest MPKI scores.

Lastly, Fig. 5 shows the accuracy of each model for all the benchmarks. The 2-level Adaptive Predictor shows a boost in accuracy in the *li* benchmark as compared to the other benchmarks, which indicates that it has a higher accuracy with increase in branches. The Bias-Free predictor has the highest accuracy of all, with 98.2% accuracy in the *perl* benchmark.

All in all, the Bias-Free predictor put up significantly better scores on the board as compared to the other predictors, including the Perceptron predictor. This implies that filtering out biased branches as well as using a recency stack-like approach to the global history register does improve the performance of the Perceptron predictor.

V. Conclusion and Future Work

This paper explored some branch prediction techniques through implementation as well as a comparative study using application benchmarks from the SPEC CPU benchmark suite. The study revealed that the Bias-Free predictor ranks higher than the other predictors in terms of IPC, MPKI and Accuracy. It is also observed that the Perceptron has a lower than expected performance as a result of lower memory budget.

While the Bias-Free architecture outperformed the other models that were considered in this study, there is still room for improvement in the branch prediction domain. The main tasks in branch prediction today are to improve performance and make predictors more energy efficient. One way to improve existing algorithms for branch prediction is to identify and eliminate nested loops, which would decrease the time and energy consumed to make predictions. A cache analysis would lead to the discovery of ways to further optimize branch prediction.

In extension to the Bias-Free predictor, there is a requirement for methods that can analyse and detect biased branches with high accuracy. Some of the existing techniques can be bettered to further improve the performance.

References

[1] Kedar Bellare, Pallika Kanani, and Shiraj Sen. "Dynamic Branch Prediction using Machine Learning Algorithms". In: (2006).

[2] Brad Calder et al. "Evidence-based static branch prediction using machine learning". In: *ACM Transactions on Programming Languages and Systems* 19.1 (1997), pp. 188–222. DOI: 10.1145/239912.239923.

[3] Prachi Chaudhary et al. "An efficient branch predictor for improved accuracy of instruction level parallelism". In: *The Journal of Supercomputing* 77.10 (2021), pp. 12098–12120.

[4] R Dhanalakshmi and C Chellappan. "Secure Email–Ensuring CIA (Confidentiality, integrity and Authentication) and Privacy for Organizations". In: ().

[5] R Dhanalakshmi and T Sri Devi. "Adaptive cognitive intelligence in analyzing employee feedback using LSTM". In: *Journal of Intelligent & Fuzzy Systems* 39.6 (2020), pp. 8069–8078.

[6] Dibakar Gope and Mikko H. Lipasti. "Bias-Free Branch Predictor". In: *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture* (2014). DOI: 10.1109/micro.2014.32.

[7] Babu Illuri and Deepa Jose. “Design and implementation of hybrid integration of cognitive learning and chaotic countermeasures for side channel attacks”. In: *Journal of Ambient Intelligence and Humanized Computing* 12.5 (2021), pp. 5427–5441.

[8] Engin Ipek et al. “Perceptron-Based Branch Prediction: Performance of Some Design Options”. In: (2005).

[9] D.A. Jimenez. “Piecewise Linear Branch Prediction”. In: *32nd International Symposium on Computer Architecture (ISCA’05)* (). DOI: 10.1109/isca.2005.40.

[10] D.A. Jimenez and C. Lin. “Dynamic branch prediction with perceptrons”. In: *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture* (). DOI: 10.1109/hpca.2001.903263.

[11] Daniel A Jiménez. “Fast path-based neural branch prediction”. In: *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE. 2003, pp. 243–252.

[12] Daniel A Jiménez. “Improved latency and accuracy for neural branch prediction”. In: *ACM Transactions on Computer Systems (TOCS)* 23.2 (2005), pp. 197–218.

[13] Daniel A. Jimnez. “Code placement for improving dynamic branch prediction accuracy”. In: *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation - PLDI ’05* (2005). DOI: 10.1145/1065010.1065025.

[14] Daniel A. Jimnez and Calvin Lin. “Neural methods for dynamic branch prediction”. In: *ACM Transactions on Computer Systems* 20.4 (2002), pp. 369–397. DOI: 10.1145/571637.571639.

[15] Chit-Kwan Lin and Stephen J. Tarsa. “Branch Prediction Is Not A Solved Problem: Measurements, Opportunities, and Future Directions”. In: *2019 IEEE International Symposium on Workload Characterization (IISWC)* (2019). DOI: 10 . 1109 / iiswc47752 . 2019 . 9042108.

[16] R Meena and V Thulasi Bai. “Study on Machine learning based Social Media and Sentiment analysis for medical data applications”. In: *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC).* IEEE. 2019, pp. 603–607.

[17] Milad Mohammadi et al. “Energy Efficient On-Demand Dynamic Branch Prediction Models”. In: *IEEE Transactions on Computers* 69.3 (2020), pp. 453–465. DOI: 10.1109/tc.2019.2956710.

[18] André Seznec. “A 256 kbits l-tage branch predictor”. In: *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)* 9 (2007), pp. 1–6.

[19] Lucian N Vintan et al. “An alternative to branch prediction: pre-computed branches”. In: *ACM SIGARCH Computer Architecture News* 31.3 (2003), pp. 20–29.

[20] Lei Zhang et al. “A Dynamic Branch Predictor Based on Parallel Structure of SRNN”. In: *IEEE Access* 8 (2020), pp. 86230–86237. DOI: 10.1109/access.2020.2992643.