2021 IEEE 8th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON) | 978-1-6654-0962-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/UPCON52273.2021.9667663

CPU Branch Prediction Using Perceptron

Akash D. Halke
Department of Electronics and Telecommunication Engineering
College of Engineering
Pune, India
halkead19.extc@coep.ac.in

Ashwini A. Kulkarni
Department of Electronics and Telecommunication Engineering
College of Engineering
Pune, India
aak.extc@coep.ac.in

***Abstract*—The branch predictors instruct the CPU, which direction is to be taken while executing branches so that the CPU will not stall fetching instructions in the pipeline. If the predicted direction is wrong, then the pipeline stalls and re-fetches the correct instructions. This penalty has to be paid by the CPU for its wrong prediction. So constantly working on reducing mispredictions is at most important. The difference between static and dynamic branch prediction is that static gives prediction at the time of compilation while dynamic gives prediction at the time of execution. Dynamic branch predictors maintain a history table that keeps a record of the last few branches' behaviors. So, improving dynamic branch prediction is of great significance as it contributes to the speed of the processor. We are attempting to simulate and analyse such a predictor using perceptron to investigate its performance as compared to counter-based predictors.**

Keywords—Branch, predictor, neural, perceptron, dynamic, static, bimodal, gshare

Introduction

Branch prediction is a notable feature of the microprocessor. Because the speed of the processor is directlyrelated to the performance of the branch predictor. Branches are instructions that can change the flow of a program's execution. These correspond to high-level language constructs like if statements and for and while loops. Branches create a performance bottleneck because, without prediction, the processor must stall until the branch completes execution andthe correct path is known .

There are mainly two kinds of branch predictors. Static prediction makes its decision depending upon some fixed set of criteria. It gives the same prediction for an instruction placed at a specific memory address. For example, all branches are taken for a memory address if it is always taken. In a loop, backward branches are taken while forward branches are not taken where backward branches always have lower addresses than their target. Dynamic prediction makes its decision by considering behaviors of the past few branches based on some mechanism. The histories of branch behavior are recorded in a branch history register or buffer. It adopts the real-time behavior of a program and accordingly produces results. Examples are bimodal, gshare, two-level adaptive predictors. The dynamic branch prediction accuracy is superior to that of static due to its quality of adaptability. The state-of-the-art branch predictors correlate between branches using long history as prediction mechanism which require more memory to store histories.These are generally counter based predictors. The accuracy of predictors mostly depend upon hardware budget. To make the predictors less dependent on size of the memory, finding new alternatives is must. Current branch prediction methods are experiencing losses and a new approach may prove more efficient.

Literature Review

Yeh and Patt [1,3] propose a two-level branch predictor (BP) which is implemented using a branch history register(BHR) table and a branch pattern history table (PHT). BHR keeps a record of branch histories, and it is indexed by lower bits of the program counter. PHT tells how likely a pattern isto repeat. PHT is indexed by BHR patterns. All historyregisters to access the same PHT so it is referred to as global PHT. The P bits of BHR can store 2p different patterns. The history pattern of the last P branches is deciding factor of branch prediction. Mcfarling [2] notes that local history BP has higher accuracy than global BP only when BP size is small. The different branches can be distinguished if BP sizeis small but in the case of larger size, aliasing of branch addresses happens. To reduce aliasing, different methods of indexing are introduced. In gselect, the lower order 4 bits of both branch address and BHR are concatenated. It reduces aliasing but cannot eliminate. So, in gshare, all 8 bits of branch address and BHR are XORed to give unique indexing to eachof the entries in the prediction table.

The predictors discussed above keep track of branch histories and predict the next direction or address of the branch depending upon the previous few branch results, but they do not learn history patterns to predict a new pattern. So, there is a need of upgrading these branch predictors to more adaptive ways. Hence, we are investigating perceptron as the basis of the prediction mechanism. Jimenez and Lin [5] attempt to implement with the same and study using traces of some programs, but the results are not supported by real-time simulation. Analysing traces and simulating real-time programs are two different things. So, we are attempting to simulate the perceptron predictor and analyse it on a virtual CPU. Mao, Zhou, Gui and Shen [13] explore deep neural networks (DNNs) to predict branches. The branch prediction is treated as a classification problem and explored both deep convolutional neural networks (CNNs) and deep belief networks (DBNs) for branch prediction. The DBNs and CNNs outperform the perceptron predictor. The experiment is concluded that deeper CNN structures lead to lower misprediction rates. Anastasios Zouzias, Kleovoulos Kalaitzidis, Boris Grot [14] attempt to explore branch prediction as Reinforcement Learning(RL) problem. In RL, the learning agent seeks to find function that maximises the reward function by interacting and learning from the environment. So it becomes easy to design considering each aspect of the predictor such as its state representation, decision-making policy and strategy to minimize the number of mispredictions.

What Is The Perceptron

Perceptron [10] is the simplest model in artificial neural networks. It categorizes the inputs with the help of a hyperplane. It also learns input patterns using the Boolean function. Perceptron comes under supervised learning type in machine learning. It accepts some input n. It contains n-1 weights and a bias. Bias is used for shifting the learning threshold. The weights are multiplied with corresponding input values then the summation of these multiplied entities is used to make the decision.

Fig. 2 shows a graphical model of a perceptron branch predictor. The input variables x_1 through x_n are branch history bits that are taken from the global branch history register (GBHR). The binary inputs are converted to bipolar so that calculations can be simplified. i.e., If GBHR entry is 0 then we consider it as -1 which means the branch is not taken and if GBHR entry is 1 then we consider it as 1 which 0 means the branch is taken. Weights w_1 through w_n are associated with their corresponding inputs from GBHR. The larger the absolute values of weights, the higher is the degree of correlation between output and branch history inputs.

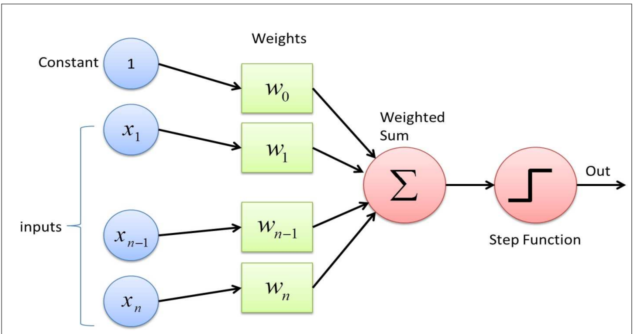

Fig. 1. Single Layer Perceptron

These values are from the weight table which is indexed by branch address. The output 'y' is calculated as the summation of products of each input bit and corresponding weight.

BRANCH PREDICTION USING PERCEPTRON

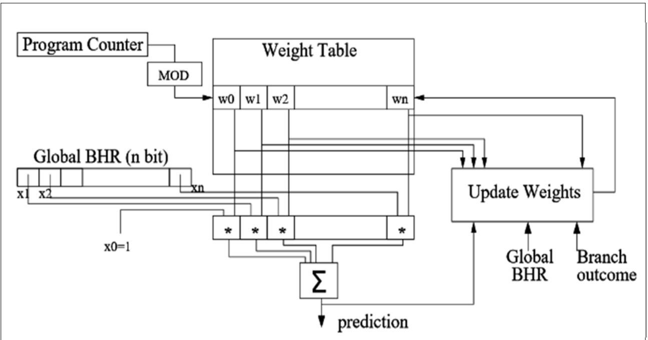

Fig. 2. Graphical Model of Perceptron Branch Predictor

A. Weight Table

The weight table is the core element of perceptron branch prediction. This body holds the weights which are responsible for learning the branch behavior. It is a 2D array in which each row is related to a different branch. Each entry in the row of the weight table decides how strongly the current branch is correlated with the entry in the global branch history register. The ith entry in the GBHR is mapped to the ith entryin the weight table. W_0 is reserved for the bias input, meant for the current branch itself which is a measure of local history. The entries in the weight table are updated wheneverthere is a change in the prediction and the actual outcome or when the prediction value is below through which means perceptron is not enough trained.

B. Global Branch History Register

The GBHR, a shift register, keeps a track record of outcomes of the last x branches. We call x as the history length. Whenever a branch is detected, GBHR stores the outcome of the branch in GBHR1, and all values in the register shifted down the array once. The value of GBHRi is the outcome of the past Ith branch. For example, GBHR3 is the outcome of the previously 3rd branch. Here, the outcomes are considered as bipolar, meaning branch 1 if the branch is taken, -1 if the branch is not taken. GBHR0 is always set to 1 providing the bias input, weight[0] to pass through the dot product unmanipulated.

C. Branch Prediction

Whenever the CPU detects a branch, the predictor accesses a row of the weight table corresponding to the branch address. Then prediction is calculated as the summation of products of each GHBR bit and corresponding weight. If it is positive, then the branch is taken and not takenfor a negative outcome. If x_i is the value at $GBHR_i$ and w_i is the value at $weight_i$, then the outcome y can be given by the simple formula :

$$y = w0 + \sum n \quad (xi * wi) \qquad .... (1)$$

Where i iterate from 0 to n.

D. Learning

A neural network [8] is well known for its ability to learn new data. For learning from the data, the weight values must be updated after the prediction is made. We update the weight table in two cases: when the value of the outcome y is belowthe threshold and when the prediction goes wrong. Each entryin the weight table is incremented if its corresponding GBHRentry is the same as the prediction and decremented if it is not. Let's say m is the outcome of the branch (1 for taken, -1 for not taken), xi is the corresponding entry in the GBHR, then the product of both be added to the existing weight table value. Because multiplying the GBHR entry and the outcomeof the branch multiplying these will result in a 1 if they agree,and a -1 if they disagree as values are bipolar. We can use these to add this value to the existing weight table entry which will increment or decrement the value appropriately. Then GBHR is shifted to accommodate new entry.

IMPLEMENTATION

A. Simplesim – 3.0

Simplesim [7] contains two simulators i.e., functional and performance simulator. The functional simulator provides the same environment as a microprocessor where program inputs are converted to outputs with the help of a supporting instruction set. But it does not give time statistics such as cycles per instruction (CPI), speed of the processor, etc. The performance simulator keeps track of each clock cycle to measure the performance of the processor. As our work needs CPI, IPC values, prediction rates for analysis of predictors thus we choose to use a performance simulator.

B. *Predictors Implementation*

All the predictors are implemented using switch case statements. We choose the branch predictor class to get its performance statistics.

Bimodal: The configuration file is changed to simulate bimodal predictor and history table size entries are varied from 2 to 4096 bits. The table size can be varied only in the powers of 2.

G-share: The configuration file is changed to simulate the gshare predictor where l1size (l1 table size), l2size (l2 table size), shift width, and XOR. Here l2 is varied from 2 to 4096bits, shift width 8 to 128, and xor status is kept as 1.

C. *Perceptron Implementation*

All code modifications are made to bpred.c on the SimpleScalar v.3.0 toolkit to accommodate the perceptron predictor. Sim-outorder.c and bpred.h are also modified to reflect the changes made in bpred.c Executables are compiled for x86 machines. A new knob perceptron is registered in sim-outorder file. It accepts 3 arguments as input that are weight table entries, size of each weight table entry and BHR size. Enum bpred_class is edited to add BpredPerceptron. Then BpredPerceptron is introduced in the bpred_create function as a case of the switch statement and parameters are passed from simoutorder through default.cfg. The weights initialization is kept to zero and all bits of the global branch history register are initialized to 1. The branch prediction mechanism is implemented in bpred_dir_lookup. The bpred_lookup function is adjusted to accommodate perceptrons then the return values of bpred_dir_lookup are used to predict a branch is taken or not. The training and update function of the perceptron predictor is implemented in bpred.c file as bpred_update function. Whenever the predictor value is below threshold or predictor mispredicted then this function monitors that and makes changes on updates accordingly to the weight table of the perceptron branch predictor.

D. *Benchmarks*

All the predictors are tested on in-built benchmarks. We use eight benchmarks namely test-printf, test-math, anagram, test-dirent, test-llong, test-lswlr, test-args, test-fmath. Test- printf has nearly 1 million instructions that are committed to be executed. The src file of test-printf contains a set of c code lines that mostly execute the print function. Test-math benchmark has nearly 60000 instructions to be committed to be executed. This benchmark performs mathematical operations. Test-fmath also performs math operations but the data type used while performing operations is a float data type. Anagram benchmark performs a combination of different arrangements of letters to form a phrase. The src file of test-llong contains printing of standard output of data type long long.

E. *Simulation*

The simplesim-3.0 simulator is configured to alpha architecture. The simulation [11] of predictors performed for each benchmark. Firstly, default.cfg file is configured to bimodal. The history table size is varied from 8 bits to 4096 bits. Thevalues CPI, IPC, prediction rate, cache hits, misses, and accesses are recorded. Then Gshare predictor is configured using default.cfg file. The parameters such as level 1 table size, level 2 table size, shift width, xor is set to 1,4096,12, 1 respectively and recorded the statistics. Then parameters varied to 1,2,8,1 in the multiples of 2. The perceptron model is also configured with parameters l1 table 128, l2 table 8, shift width 27, and recorded the statistics of the predictor.

RESULTS

Bimodal and gshare predictor shows highest values of CPI. In the case of perceptron, CPI values are lowest as shown in Table 1 which means that there is a significant drop in cycles required to execute an instruction if perceptron predictor is used. The address prediction rates of perceptron predictor are highest while prediction rates of gshare and bimodal are also approaching that of the perceptron. Due to long history and weight table size, the address prediction rate improves with the increase in history length (fig. 3). Gshare and bimodal which are counter-based predictors do not perform with perceptron which is a learning-based predictor. CPI values and address prediction rates from tables show that perceptron predictors perform better than other predictors. As history length increases, there is significant decline in the cache accesses in all three cases. While among bimodal, gshare and perceptron predictors, bimodal has most and perceptron has least number of cache accesses which also shows that our perceptron model is working efficiently. If any predictor gives less mis-predictions, less number of instructions executed because of not having penalties. The perceptron outperforms bimodal and gshare predictors clearly. One weakness of perceptron predictors is their hardware complexity as compared to counter-based predictors.

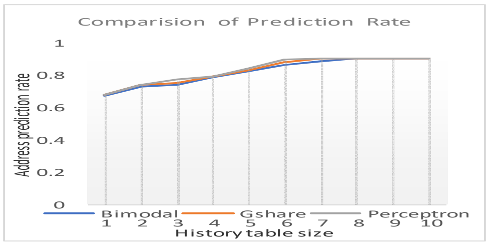

Fig. 3. Performance Statistics of Predictors for 'Test-Printf' Benchmark

CONCLUSION

Perceptrons are great pattern learners and seemed convenient for performing the CPU branch prediction. The counter-based predictors predict branches just using a fixed mechanism which cannot perform well in some scenarios of branch execution but perceptrons can learn the behavior of branches using history patterns. The cycles per instruction value significantly reduced in comparison with the counter- based predictor and also when history length is increased.

One weakness of neural-based perceptron predictors is that the hardware required for the computation of prediction is more complex than counter-based predictors. But improved prediction rate and CPI values of perceptron attract more work on neural network-based methods. Though perceptron-based predictor is performing well in comparison with counter-based predictor, there is room for future work. Other machine learning methods such as convolutional neural networks, multilayer perceptron, decision tree, etc. can be implemented to improve the performance of dynamic branch predictors constantly.

TABLE I. PERFORMANCE STATISTICS OF PREDICTORS FOR 'TEST-PRINTF' BENCHMARK

History Table Size (in bits)		***8***	***16***	***32***	***64***	***128***	***256***	***512***	***1024***	***2048***	***4096***
Bimodal	***Cache Accesses***	1413115	1336824	1316254	1317163	1276477	1241435	1211202	1190663	1189804	1189804
	Cache Hits	1371830	1296079	1277677	1278579	1237732	1202729	1172863	1153106	1152222	1152222
	Cache Misses	41285	40745	39325	38584	38745	38706	38339	37557	37582	37582
	Address Prediction rate	0.6751	0.7283	0.743	0.7878	0.8241	0.8635	0.8855	0.9023	0.9031	0.9031
	Total Instruction s Executed	1279485	1240034	1206541	1199068	1170622	1145342	1122357	1107373	1106586	1106586
	IPC	1.1381	1.1879	1.2342	1.2804	1.3354	1.3945	1.4345	1.4637	1.4645	1.4645
	CPI	0.8786	0.8418	0.8102	0.781	0.7488	0.7171	0.6971	0.6832	0.6828	0.6828
Gshare	***Cache Accesses***	1400004	1392245	1381546	1276541	1224568	1211001	1189804	1189804	1189804	1189804
	Cache Hits	1360759	1353118	1342652	1237787	1185843	1172545	1152222	1152222	1152222	1152222
	Cache Misses	39245	39127	38894	38754	38725	38456	37582	37582	37582	37582
	Address Prediction rate	0.6771	0.7394	0.7527	0.7896	0.8321	0.8812	0.9031	0.9031	0.9031	0.9031
	Total Instruction s Executed	1265125	1242036	1206354	1186527	1164529	1142345	1106586	1106586	1106586	1106586
	IPC	1.2297	1.2632	1.2883	1.3469	1.4033	1.4467	1.4645	1.4645	1.4645	1.4645
	CPI	0.8132	0.7916	0.7762	0.7424	0.7126	0.6912	0.6828	0.6828	0.6828	0.6828
Perceptro n	***Cache Accesses***	1398754	1345666	1395642	1236852	1175879	1172465	1171032	1171032	1171032	1171032
	Cache Hits	1359700	1306841	1357169	1198484	1137754	1134707	1133450	1133450	1133450	1133450
	Cache Misses	39054	38825	38473	38368	38125	37758	37582	37582	37582	37582
	Address Prediction rate	0.6794	0.7421	0.7727	0.7912	0.8432	0.8964	0.9035	0.9035	0.9035	0.9035
	Total Instruction s Executed	1231215	1216215	1192369	1163245	1141237	1108654	1104312	1104312	1104312	1104312
	IPC	1.2714	1.346	1.3861	1.3978	1.4357	1.4679	1.492	1.492	1.492	1.492
	CPI	0.7865	0.7429	0.7214	0.7154	0.6965	0.6812	0.6702	0.6702	0.6702	0.6702

REFERENCES

[1] T.-Y. Yeh and Y. Patt, “Two-level adaptive branch prediction,” In Proceedings of the 24th ACM/IEEE Int’l Symposium on Micro-architecture, November 1991.

[2] S. McFarling, “Combining branch predictors,” Technical Report TN-36m, Digital Western Research Laboratory, June 1993.

[3] T.-Y. Yeh and Y. N. Patt, “An alternative implementation of two-level adaptive branch prediction,” ISCA,1992, pages 124-134.

[4] Michaud and Senzec, “A comprehensive study of dynamic global history branch prediction,” Internal Publication, 2001.

[5] D. A. Jimenez and C. Lin, “Dynamic Branch Prediction with Perceptrons,” Proceedings of the 7th International Symposium on High-Performance Computer Architecture, January 2001.

[6] Smith, “A study of branch prediction strategies,” IEEE, 1981.

[7] Simplesim-3.0. http://www.simplescalar.

[8] Simon Haykin, “Machine Learning,” Pearson Education Ltd, 2009, pages 47-54.

[9] Sparsh Mittal, “A survey of techniques for dynamic branch prediction,” Concurrency and Computation: Practice and Experience, 2019.

[10] H. D. Block, “The perceptron: a model for brain functioning,” Reviews of Modern Physics, vol. 34: pages 123-134, 1962.

[11] G. H. Loh., “Simulation Differences Between Academia and Industry: A Branch Prediction Case Study,” IEEE International Symposium on Performance Analysis of Systems and Software, 2005.

[12] P.-Y. Chang, E. Hao, and Y. N. Patt, “Alternative implementations of hybrid branch predictors,” Proceedings of the 28th Annual International Symposium on Microarchitecture, pages 252 – 257, Dec. 1995.

[13] Y. Mao, H. Zhou, X. Gui and J. Shen, “Exploring Convolution Neural Network for Branch Prediction,” in IEEE Access, vol. 8, pp. 152008-152016, 2020.

[14] Anastasios Zouzias, Kleovoulos Kalaitzidis and Boris Grot,”Branch Prediction as a Reinforcement Learning Problem,” arXiv:2106.13429v1, 25 Jun 2021.