Journal of Instruction-Level Parallelism 9 (2007) 1-18 Submitted 4/07; published 5/07

A^3PBP: A Path Traced Perceptron Branch Predictor Using Local History for Weight Selection

Yasuyuki Ninomiya Y-NINO@CACAO.CS.UEC.AC.JP
Kôki Abe ABE@CACAO.CS.UEC.AC.JP
Department of Computer Science, The University of Electro-Communications
1-5-1 Chofugaoka Chofu-shi,Tokyo 182-8585 Japan

Abstract

We present a new perceptron branch predictor called Advanced Anti-Aliasing Perceptron Branch Predictor (A^3PBP), which can be reasonably implemented as part of a modern microprocessor. The predictor has two important features: (1) local history is used as part of the index for weight tables, and; (2) execution path history is used effectively. In contrast with global/local perceptron branch predictors, where a pipeline architecture cannot be used, the scheme enables accumulation of weights to be pipelined and effectively reduces destructive aliasing, which in turn significantly increases prediction accuracy. Our scheme has lower computational costs than Piecewise Branch Predictor and utilizes execution path history more efficiently than Path Trace Branch Predictor, two well-known perceptron branch predictors. We present a version of A^3PBP in which the number of pipeline stages is reduced as much as possible without increasing critical path delay. The resulting predictor is very accurate. We also discuss how implementable the presented predictor is with respect to computational latency, memory requirements, and computational costs.

1. Introduction

In recent years, central processing units have tended to become more deeply pipelined. The deeper the pipeline, the shorter the cycle time, and the more each branch misprediction degrades performance.

Perceptron branch predictors have been extensively studied in recent years in an attempt to reduce misprediction rates [1]. Because improving branch prediction significantly improves the performance of the processor, branch prediction research is potentially very rewarding. In this paper, we propose a new perceptron branch predictor called Advanced Anti-Aliasing Perceptron Branch Predictor (A^3PBP) that can be reasonably implemented as part of a modern microprocessor. The behavior of a branch can be accurately predicted based on its past behaviors with respect to the execution path history up to the branch [2]. However, a branch's behavior may differ from its execution path history in some cases, and, when this occurs, predictors that use path history alone fail to accurately predict the outcome of the branch. This phenomenon is called destructive aliasing. To resolve the destructive aliasing problem, we propose using the local history of each branch of the execution path history. Our scheme modifies the address of each branch on the execution path history by replacing a part of the address with its local history and indexes the weight table using the modified execution path history. Therefore, when the scheme predicts the behavior of a branch, it can use individual weight table entries in situations where the execution paths

are initially the same but are modified to be different. This enables the scheme to resolve the destructive aliasing problem.

Unlike the conventional global/local perceptron branch predictor, which cannot be pipelined, our scheme enables accumulated weights to be pipelined. Our scheme has lower computational costs than Piecewise Branch Predictor [3] and utilizes execution path history more efficiently than Path Trace Branch Predictor [4], two well-known perceptron branch predictors.

This paper is structured as follows. Section 2 describes recent studies on perceptron branch predictors, focusing on how they were improved and problems still to be overcome. Section 3 proposes a new perceptron branch predictor architecture as a solution to these problems. Section 4 evaluates A^3PBP and compares its evaluation results with those for other predictors. Section 5 describes a version of A^3PBP that uses fewer pipeline stages without increasing the critical path delay. The results of the experiment with this architecture are also presented. Section 6 discusses how implementable the presented predictor is with respect to computational latency, memory requirements, and computational costs. Finally, we draw some conclusions in Section 7.

2. Related Work

2.1. Global/Local Perceptron

Hereafter, we refer to the current branch whose outcome is to be predicted as branch B. The perceptron branch predictor proposed by Jimenez [1] uses as its inputs the branch B addresses and the execution results of branches in the program in a time sequence that constitutes the global or local history. It reads a weight vector from a weight table, using the branch B address as the index. The element of the weight vector corresponding to the element of the branch history uses values of 1 or -1 to indicate taken or untaken. Let W_i be the weight and X_i be the element of the corresponding branch history, $1 \leq i \leq n$. Calculate value y by the following equation.

$$y = W_0 + \sum_{i=1}^{n} W_i X_i,$$

where W_0, the bias, is read together with the weight vector. The predictor outputs the value 1 if $y > 0$ and 0 otherwise.

The weights are updated by the equation:

$$W_i = W_i + \alpha t X_i$$

where branch result t belongs to $\{1, -1\}$ and α is set to 1. The perceptron does not usually update the weights when it succeeds in distinguishing the patterns. However, in the perceptron branch predictor the weights are updated even if the prediction was correct when the updated value is less than a certain threshold.

Although the Global/Local Perceptron predicts more accurately than conventional predictors such as gshare [5], its prediction latency is long because its calculations are very computationally complex. Therefore, the scheme cannot be practically implemented on a CPU.

2.2. Path-Based Neural Branch Predictor

Because of its pipeline structure, the Path-based Neural Branch Predictor [2] has a shorter latency than the Global/Local Perceptron Branch Predictor. It uses execution path history instead of the branch B address as the index to read the weights. The branch B address is only used to read the bias. The weights are accumulated in pipelined fashion before the branch B address is inputted, avoiding the latency issue of the Global/Local Perceptron.

However, when the program changes execution patterns, the path history it uses for prediction is incorrect, resulting in less accurate prediction than the Global/Local Perceptron Branch Predictor, even if the length of the history is increased.

2.3. Piecewise Linear Branch Predictor

Idealized Piecewise Linear Branch Predictor [6] uses the branch B address as well as the path, i.e., a dynamic sequence of branches leading to B, as an index to access weight tables. It uses an XOR of the branch B address and the n-th oldest address within the path as the index to access weights for the n-th global branch history. The scheme predicts more accurately than the Global/Local or Path-based predictors. However, using the address of the current branch B as the index to access every weight table does not allow it to accumulate the weights before the branch is fetched. The ability to accumulate the weights in advance is essential to using a pipeline structure to reduce latency. Thus, Idealized Piecewise Linear Branch Predictor cannot be pipelined unless all branches that can possibly be fetched are exhausted in advance.

Ahead Pipelined Piecewise Linear Branch Predictor [3] which uses all possible values of the reduced address of branch B as indices for accessing weight tables upstream from the pipeline structure attempted to solve this problem. However, the computational cost of exhausting possible branches is very high, even if limiting the width of the branch B address reduces the number of cases.

2.4. Path Trace Branch Predictor

Path Trace Branch Predictor (PTBP) [4] has a pipelined structure with weight tables that are accessed in parallel. A weight table in a stage is accessed by a hashed value of addresses of branches at intervals of pipeline depth within the path leading to branch B. The scheme attempts to utilize the path information as much as possible within a pipelined structure. However, learning in PTBP tends to be too sensitive to path history, making the memory used by the weight tables inefficient.

3. Advanced Anti-Aliasing Perceptron Branch Predictor

In this section, we propose a new branch predictor, Advanced Anti-Aliasing Perceptron Branch Predictor, called A^3PBP, or A3PBP. Figure 1 shows the structure of A^3PBP, and the following explains its characteristics.

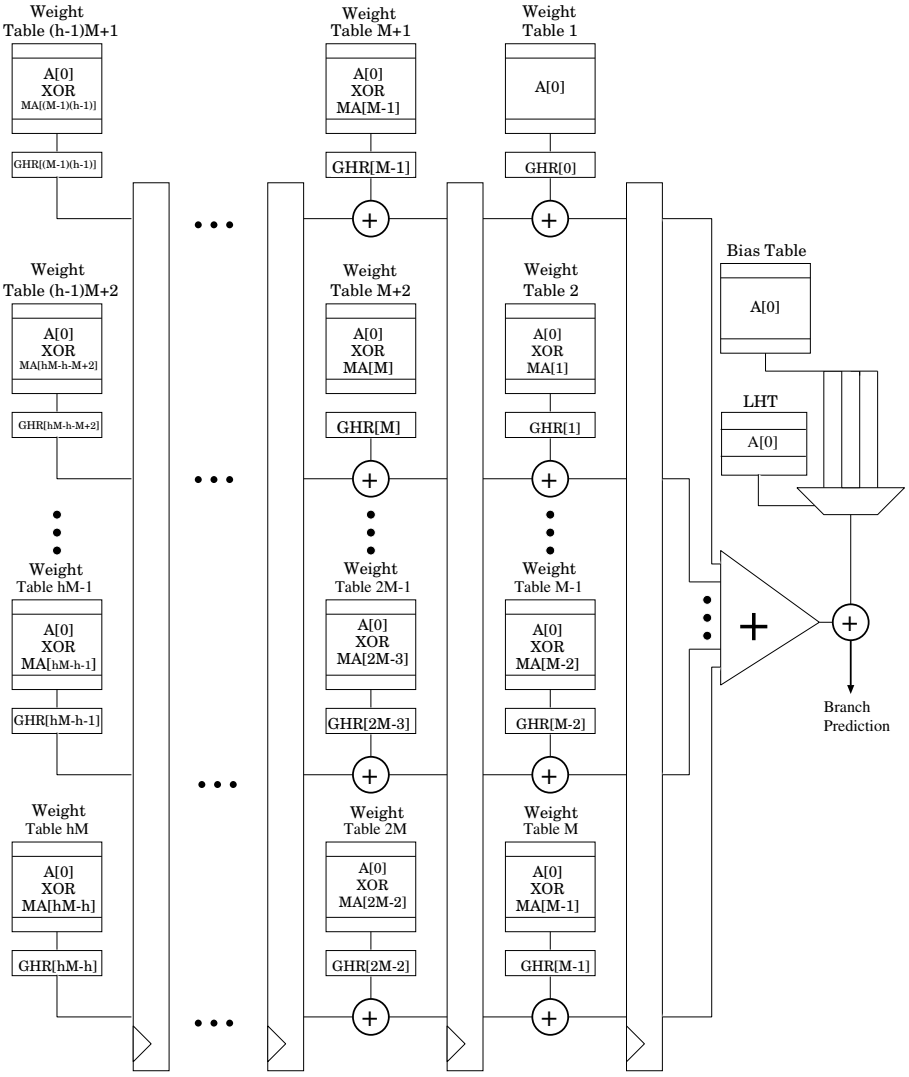

Figure 1: Structure of A^3PBP.

3.1. Utilization of Local History as Part of Index

As described in [2], the information of the path history is naturally reflected to the weights using the address of the branch whose outcome is to be predicted at the stage furthest downstream (stage 0) as the index to access weight tables in upstream stages of the pipeline structure. Here our scheme incorporates the local history of each branch on the execution path to the branch address. This enables the scheme to resolve the destructive aliasing without facing the difficulty of pipelining the prediction.

The accuracy of the Global/Local Perceptron Branch Predictor is viewed as evidence that using the local history improves prediction accuracy. However, implementable perceptron branch predictors, such as the path-based predictor, have not used local history. This is because the weights for local history of branch B, whose outcome is to be predicted,

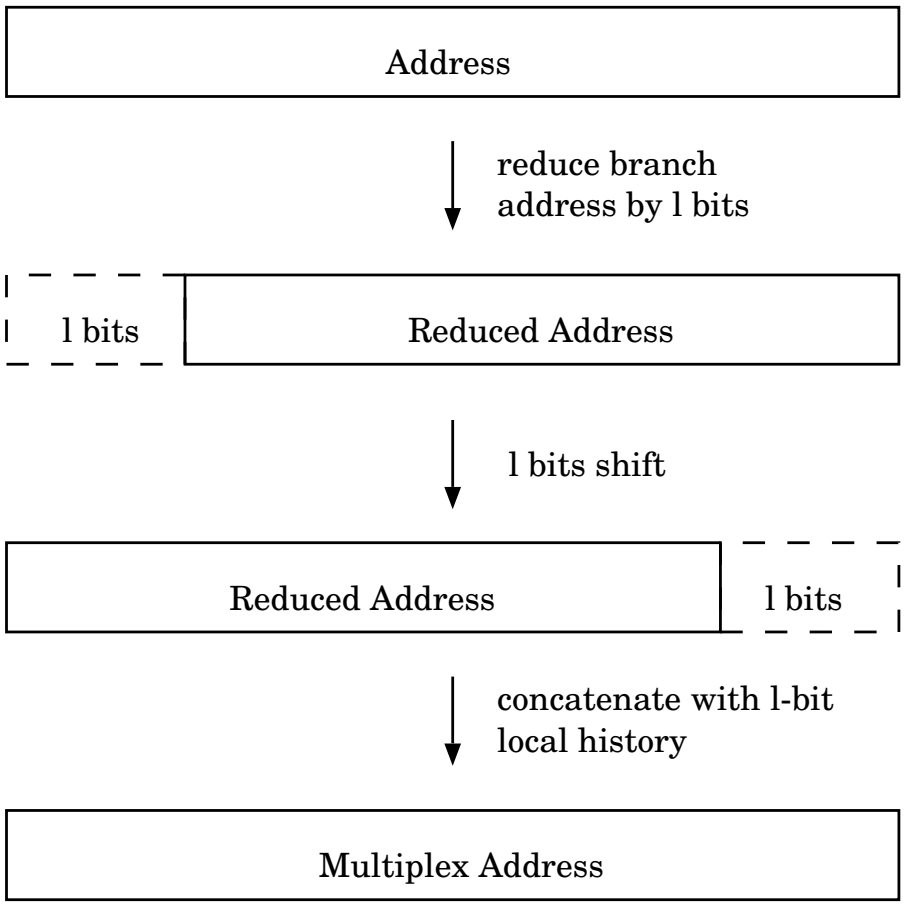

Figure 2: Formation of index MA (multiple address) for accessing weight tables.

should be accessed and accumulated at the stage farthest downstream, which will lead to a significant increase in latency. If we want to reduce latency, the weights for local history need to be accumulated along the pipeline stream. However, the index values for accessing weight tables for local history cannot be determined upstream from the pipeline if the local history of the current branch B is used as the index.

Path-Based Neural Predictor predicts branches using a pipelined structure by accessing weight tables using addresses in the path leading to branch B. If the predictor further wishes to use the local history to improve prediction accuracy, it needs to accumulate weights for local history after obtaining the branch B address. The needs does not allow the predictor to be pipelined because there is no way of accessing the local history without using the branch B address itself.

Here we propose a novel way of utilizing local history that enables prediction using a pipelined structure. Instead of preparing a dedicated weight table for local history, our scheme reflects the local history information to ordinary weight tables. For that purpose, we form the index for accessing the weight tables using a function of the branch B address and its local history. Figure 2 shows an example of the function, where the index of weight

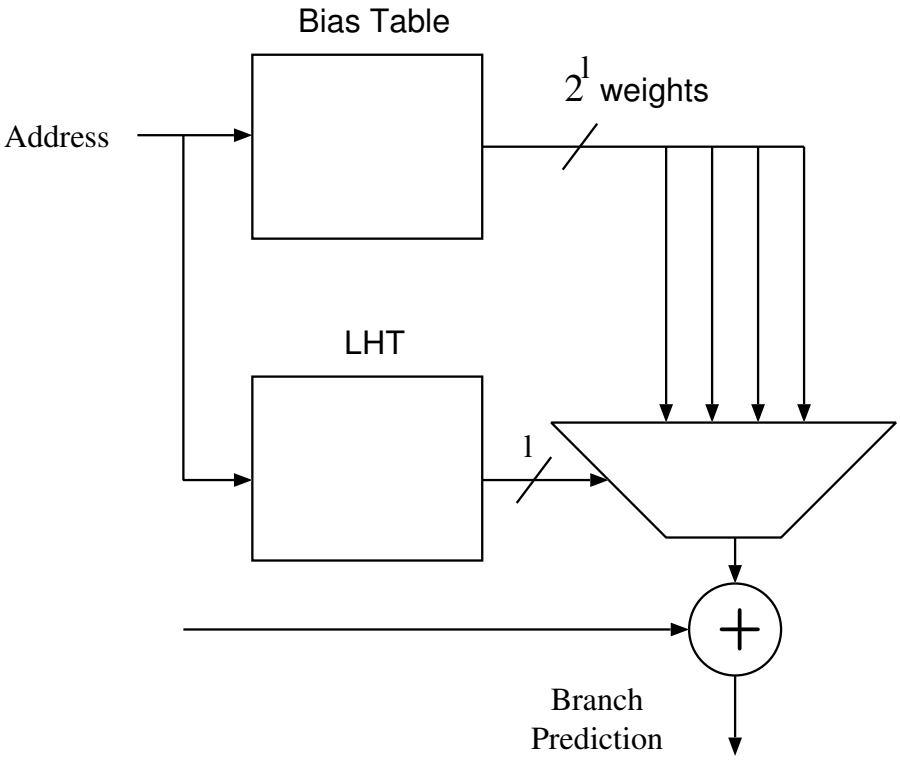

Figure 3: Structure for reading bias weights and local history in parallel.

tables is formed by reducing the branch address by l bits and concatenating it with l-bit local history read from a table called LHT (Local History Table). The overhead required for the formation of the index is negligible. Hereafter, indexes formed in this way are referred to as MAs (multiplex addresses).

In our scheme, addresses constituting the path leading to branch B are replaced with MAs, which are used to access both weights and bias weight. Indexing weight tables with MAs enables the predictor to learn multiple cases (2^l cases for the example given above) depending on the pattern of local history. Conventional predictors do not discriminate between paths consisting of the same addresses with different patterns of local history. This increases the number of destructive collisions of weights.

To implement the scheme, the bias weight must be read in parallel with the local history. This can be done by reading a set of possible bias weights in parallel with the local history and then selecting one of the bias weights using a selector, as shown in Figure 3. This minimizes the increase in latency.

Here we address the issue of how to properly determine length l of the local history. To resolve the issue, we conducted an experiment to examine the dependence of prediction accuracy on the length of the local history. In the experiment, we used the trace data obtained by executing 12 SPEC CINT2000 benchmark programs of a hundred million steps using the SimCore simulator [7].

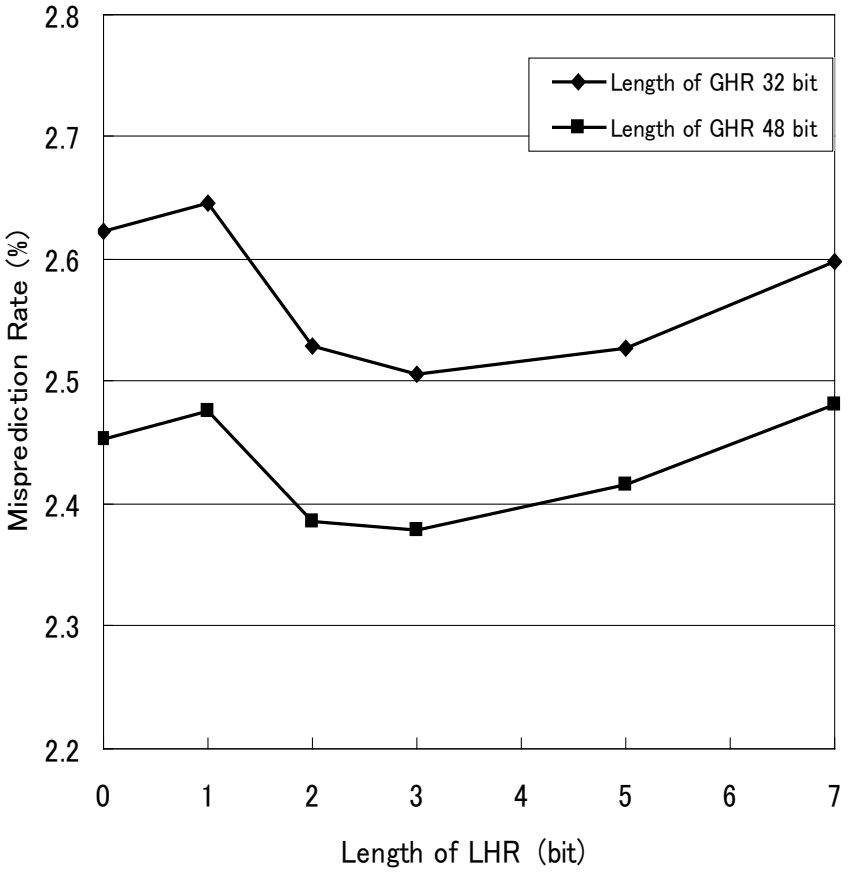

Figure 4: Misprediction rate vs. local history length.

The results shown in Figure 4 reveal that the misprediction rate is the smallest for length $l = 3$ though it is almost the same as with $l = 2$. Because the longer l increases the number of weights required for parallel reading, we used $l = 2$. In this case, there are four bias candidates that facilitate the implementation. It is worth noting that the address of branch B is used to access the bias weight table. This implies that the size of bias table is 2^l times larger than that of the other weight tables. This is reasonable because a larger bias weight table helps decrease the possibility of destructive collisions of weights [8].

3.2. Effective Utilization of Execution Path History

Idealized Piecewise Linear Branch Predictor, which uses the path and the branch B address to access weight tables, is more accurate than the predictors discussed above. However, computing predictions corresponding to all possible branch B addresses with its pipelined version is very costly. When the bit length of the branch B address is reduced to suppress costs, as was done by the Ahead Pipelined Piecewise Predictor, the information contained in the execution path history cannot be fully utilized.

From the above observations, requirements for indexing weight tables, which enables effective utilization of the execution path history in pipelined architecture, are derived as follows:

- The index should be closely related to the address of branch B;
- The index should be available before the address of branch B becomes known;
- The index should have as wide a range of domain as the address of branch B.

As shown by the first and second observations, the information about the address of branch B should be extracted from the execution path history. Addresses of branches that were executed shortly before branch B should contain the information about the branch B address itself, as they do in loops.

3.2.1. Extraction of Information about Branch B Address

To demonstrate that addresses of branches that were executed shortly before branch B should contain the information about branch B address, we conducted an experiment in which we tried to predict the branch B address using the execution path history. Let X be the address of the branch that is executed d instructions after a branch (say branch A) and Y be the address of the branch that was executed d instructions after the branch A. In the experiment, we measured $P(d)$, the probability that $X = Y$. The experimental conditions are the same as those described in Section 3.1.

The results shown in Figure 5 indicate that the addresses of branch instructions closer to the branch B address on the execution path are more closely related to the address of branch B. The relation becomes weaker at smaller rates as the distance from the branch B address increases. These results show that branch addresses with distances of less than eight bits have sufficient relationships with the branch B address because the prediction rate is over 50%.

As shown in Figure 1, our architecture uses the address of branch B, whose outcome is to be predicted at stage 0 to form the index. The address is denoted by A[0]. For upstream stages of the pipeline, the value of A[0] is regarded as the address of a branch that was executed before the branch whose behavior is going to be predicted at those stages. As described above, the branch that was executed before the branch whose behavior is going to be predicted at those stages is closely related the address of branch B. This is why our architecture uses A[0] in upstream stages of the pipeline to form the index.

Here we note that in Figure 5 addresses with distances of less than about four bits, where the rate of prediction decreases abruptly, are considered to have strong relations with the branch B address. This implies that the pipeline should be shallow. This results in a two-dimensional pipeline structure of h stages with a small h value. At each stage, M accumulations are carried out in parallel. hM is the total number of weights to be accumulated, which is equal to the length of the global history. As shown in Figure 1, the index of the weight table at row m and column n of the array structure is given by a hashed value of A[0] and MA$[(n-1)M+m-n]$, where $1 \leq m \leq M$ and $1 \leq n \leq h$.

Figure 1 shows an example of A^3PBP architecture that uses XOR for the hashing. Each weight table in a stage is accessed by an index containing information about the path most

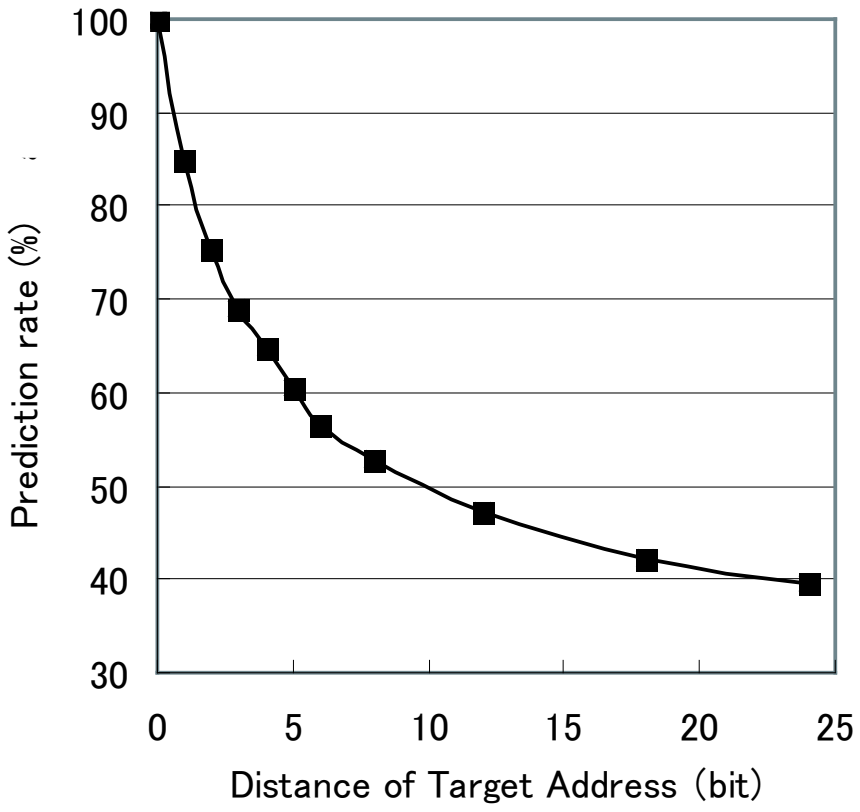

Figure 5: Prediction rate of branch B address vs. distance between branch B address and address used for prediction of execution path history.

probably related to branch B available at that stage. (The index of the weight table at row 1 and column 1 is given by A[0] XOR MA[0]. However, MA[0] is not used in the figure. This is because the local history required to compose MA[0] cannot be supplied at stage 1 without increasing the latency.)

The implication that the pipeline should be shallow was confirmed by an experiment that simulated the function of A^3PBP on the same trace data as used in Section 3.1. It did this by varying the value of h while keeping the value of hM (the length of the GHR) fixed. The results are shown in Figure 6.

These results show that the smaller the pipeline length, the more accurate the prediction. Also, the difference between the values $h = 1, 2, 4$ is small, but the prediction accuracy for $h = 8$ is significantly different from other cases. This means that, with respect to prediction accuracy, any value of $h = 1, 2, 4$ suffices but that $h = 4$ is preferable considering the latency of weight accumulation.

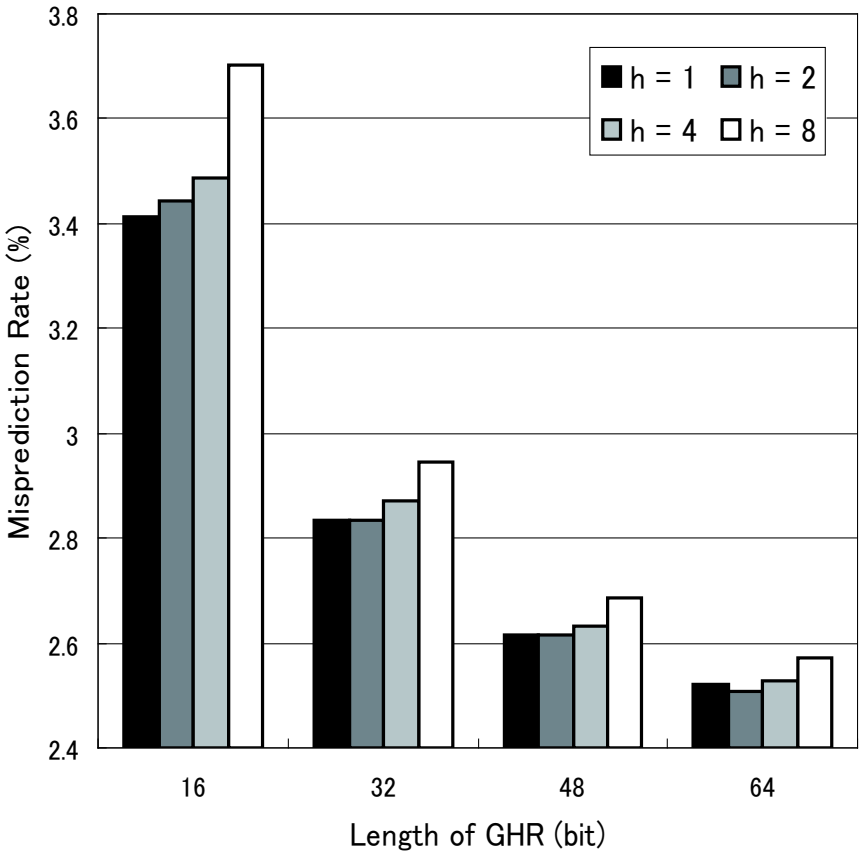

Figure 6: Misprediction rate vs. pipeline length. ($h + 1$ represents number of pipeline stages.)

3.2.2. Representing Execution Path Information

A previous study [4] uses address history instead of the branch B address to access the weight tables. PTBP predicts more accurately than the Piecewise predictor because, in addition to utilizing the execution path, it also utilizes the information of the branch B address contained in the address history. However, PTBP does not utilize path information in the optimum way. PTBP generates hashed values by XORing all of the rotated branch addresses that constitute the path and uses them as indices to access weight tables. Therefore, when an address that constitutes the path changes, indices for all the weights generated by the path change.

Figure 7 shows execution path history as used for index generation by (a) our scheme and (b) PTBP. In our scheme, the two execution paths (B_0, B_1, B_2, and B_3) and (B_0, B_{1a}, B_2, and B_3) are distinguished by the information represented by pairs (B_0,B_1), (B_0,B_{1a}), (B_0,B_2), and (B_0,B_3). In PTBP, on the other hand, exact information about the two paths is used to access weights. Although the slight change in the path can affect prediction, the

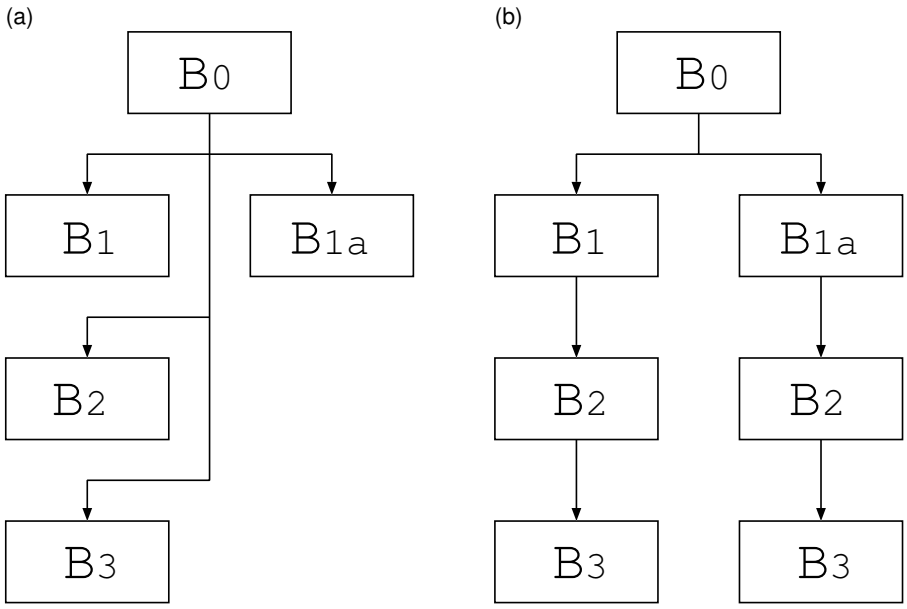

Figure 7: Execution path history as used for index generation: (a) by proposed method, (b) by PTBP.

excessive use of path information greatly increases the number of weights used to predict a branch instruction. As a result, the limited memory capacity of weight tables induces the destructive collision of weights (aliasing), which decreases prediction accuracy. Instead of using seven weights corresponding to the path, as in Figure 7 (b), we use five weights corresponding to the paths, including the most recent branch address, as shown in Figure 7 (a). Although our scheme does not reflect the path information representing relations such as (B_1,B_2), (B_1,B_3), and (B_2,B_3), prediction accuracy is improved because the excess use of the paths is suppressed and the aliasing can be avoided when the table capacity is limited.

Our scheme's use of path history is different from that of pipelined PTBP in another respect. Our scheme accesses the weight tables using information very closely related to that available for branch B at that stage, whereas PTBP accesses weight tables in one stage using hashed values of addresses at intervals of h within the path leading to branch B. The addresses used to generate the indices are located far from branch B in PTBP, resulting in failure to effectively extract the information about the branch B address from the execution path history.

There is a possibility that speculative prediction occurs before the global history has been updated when a sequence of branch instructions is contiguously executed in a program. The more speculative a prediction is, the less accurate it is. To reduce this possibility, the execution path history close to the branch B address should be used downstream from the pipeline. This is why we generate the index by A[0] XOR MA$[(n-1)M+m-n]$ to access the weight table at row m and column n of the array structure.

Table 1: History length of predictors.

Budget	Global /Local	Path Based	Piece-wise	Pipe. PTBP	A^3PBP	A^3PBP w/o LH	A^3PBP w/o EUPH
8 KB	34/12	32	20	22	16	22	32
16 KB	38/14	34	24	26	26	26	36
32 KB	40/14	34	26	32	36	32	48
64 KB	50/18	37	43	43	43	43	52
128 KB	60/20	40	50	50	63	50	68
256 KB	70/20	42	51	51	63	51	72

The bias weight is multiplied by constant c before being summed with the accumulated weights. This increases the effect of bias weight on prediction. Constant c is set to a power of two, and the multiplication is easily done by bit shifts.

4. Prediction accuracy of A^3PBP

We evaluated the prediction accuracy of A^3PBP by simulation. We also separately evaluated the effects of utilization of local history (LH) as part of an index and the effective utilization of execution path history (EUPH). We refer to the former and latter versions of A^3PBP as A^3PBP w/o EUPH and A^3PBP w/o LH, respectively. We also evaluated the Global/Local Perceptron Branch Predictor, the Path-Based Neural Branch Predictor, the Ahead Pipelined Piecewise Linear Predictor, and the Pipelined PTBP with 16 chains. The experimental conditions were the same as those described in Section 3.1. Misprediction rates were measured at memory capacities ranging from 8 KB to 256 KB. The lengths of global history used for the memory capacities are shown in Table 1. The values for A^3PBP and its derivative versions were determined experimentally. The values for other schemes were determined according to Jimenez [3] for Global/Local, Path-Based, and Piecewise and according to Ishii and Hiraki [4] for Pipelined PTBP and are given in Table 1. We used 8-bit weights except with A^3PBP with memory capacities of 8 KB, 16 KB, and 32 KB, where we used 7-bit weights. The number of bits used to shift the bias of A^3PBP was set to two for 8 KB and 16 KB and three for others. The value of M was set to the number of rows of the two-dimensional pipelined structure shown in Figure 1, which is 16. The threshold for updating weights was set to $2.1 \times (GHR + 1)$.

The simulation results of the misprediction rates of A^3PBP and other schemes averaged over 12 benchmarks are compared in Figure 8. As can be seen, A^3PBP's misprediction rate is the lowest among the predictors evaluated. It is less than that of PTBP with 16 chains by 5.3% on average and 6.6% at most with a 32 KB memory. These results imply that A^3PBP utilizes the weight tables more effectively than any other branch predictor. The misprediction rate of A^3PBP with a memory capacity of 256 KB for each of the SPEC CINT 2000 benchmark programs is compared with those of other schemes in Figure 9.

Figure 10 shows the misprediction rates of A^3PBP and its two derivative versions, A^3PBP w/o LH and A^3PBP w/o EUPH. As can be seen, the effects of LH (utilization

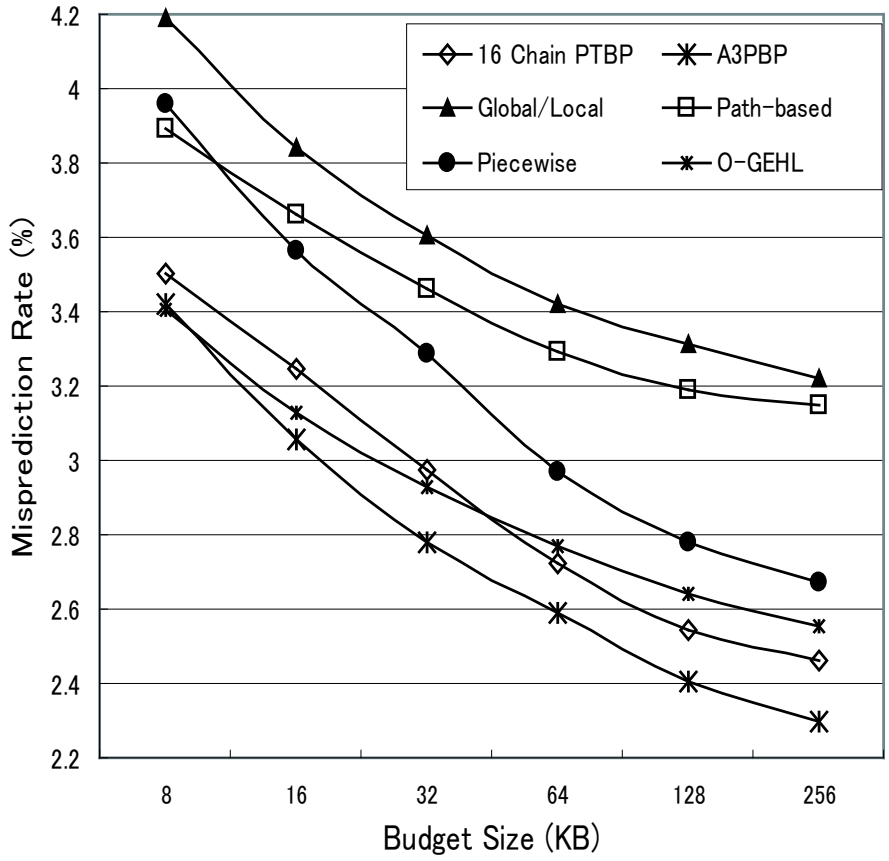

Figure 8: Misprediction rate of predictors.

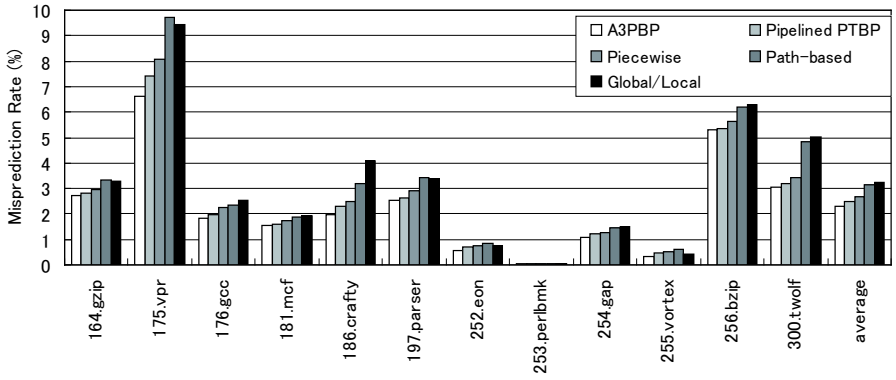

Figure 9: Misprediction rate of A^3PBP for each SPEC CINT 2000 benchmark program.

of local history as part of index) and EUPH (the effective utilization of execution path history) are comparable. The effects of LH and EUPH are also compatible, i.e., they can be applied simultaneously without interfering with each other.

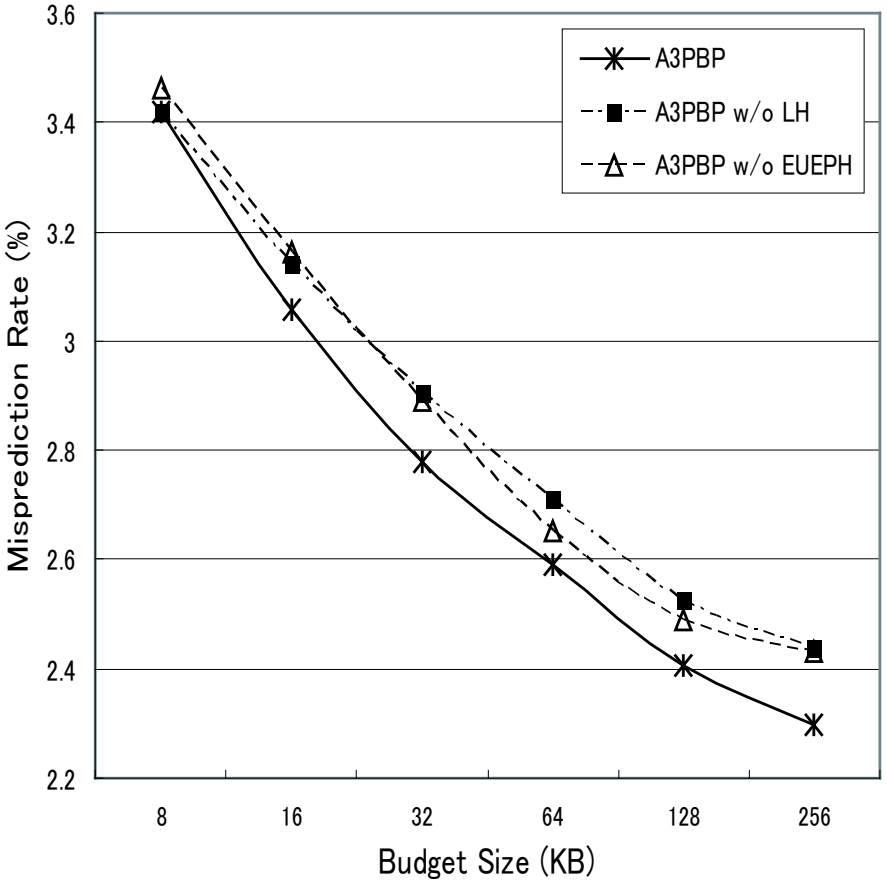

Figure 10: Effects of using path history effectively.

5. Customized Version of A^3PBP for CBP-2

Figure 11 shows a version of A^3PBP customized for 32 KB storage. There are several ways of representing the local history of a branch, e.g., using states of an automaton modeling the branch and using its Taken/Untaken patterns. Our presentation in CBP-2 [9] used the automaton of the bimodal predictor to represent the local history. However, Taken/Untaken patterns representing the local history was found to yield more accurate prediction than the automaton. Therefore, in this paper we present the results obtained using the shift register.

As we saw in Section 3, with respect to prediction accuracy, any value of $h = 1, 2, 4$ can be used. However, when the budget is severely limited, there are significant differences in misprediction rates for $h = 1, 2, 4$. Here we use the smallest value of h to obtain the A^3PBP version with the best prediction accuracy when the budget is limited to 32 KB. That is, in this customized version, only one pipeline stage (stage 1) precedes stage 0, where the outcome of the branch B is predicted. The length of the global history, hM, was set to 31, and the global history was stored in a shift register GHR (global history register). Each weight table has 1024 7-bit entries except for stage 0, where a composite table consisting

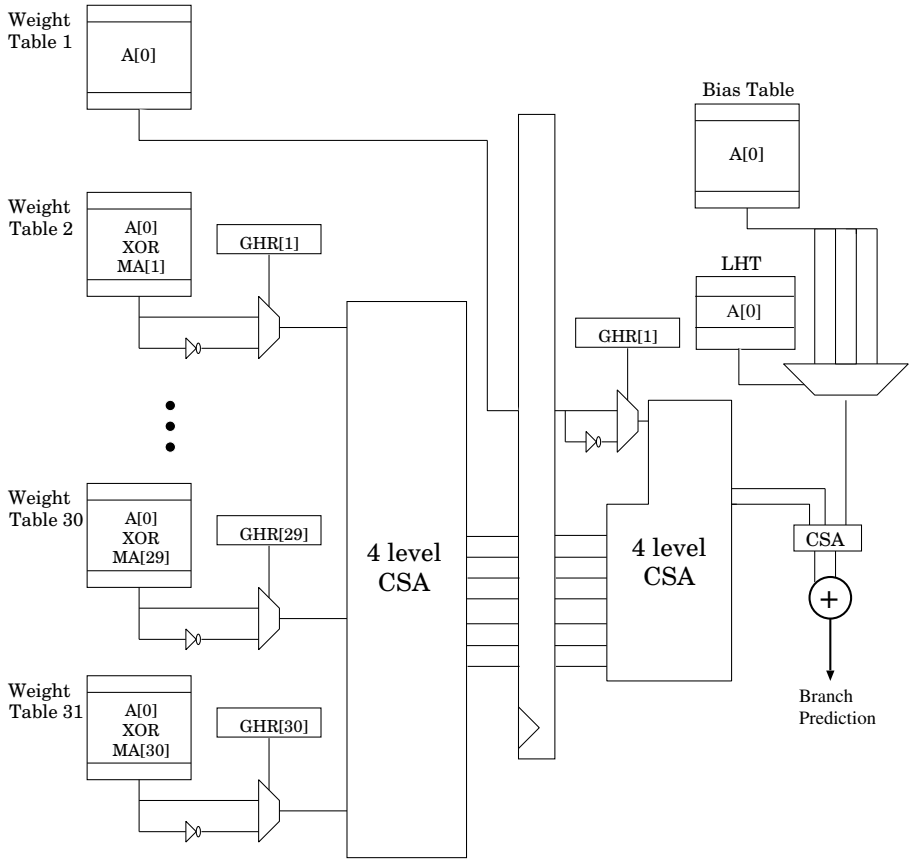

Figure 11: Structure of customized A^3PBP.

of four weight tables of the same size is used. One of the four weights from the composite table is selected by $l = 2$-bit local history read from an LHT of 4096 entries.

Weights were accumulated by a carry save adder (CSA) tree except for the addition to obtain final results at stage 0, where a carry propagation adder (CPA) is used. The CSA tree of four levels at stage 1 reduces 30 addends to seven intermediate sums that are further reduced to two sums together with another addend by a four-level CSA tree at stage 0. We replace two's complement subtraction of weights for untaken history of GHR with addition of inverted weights, leaving corrections for two's complement subtraction unapplied. This is because adding 1 for the collections causes the latency of accumulation to increase. The threshold parameter for learning was experimentally set to 79. The prediction accuracy results simulated on the set of distributed benchmarks for CBP-2 competition are reported in Table 2. The average misprediction rate is 3.897, which is 15.6% less than that of Path-Based Neural Predictor.

6. Discussions on Hardware Budget Requirements and Latency

We first discuss prediction latency for A^3PBP. The latency for stage 1 is estimated to be less than that for stage 0 based on the following discussions. For both stages critical paths

Table 2: Misprediction rates of customized version of A^3PBP for CBP-2 competition.

BenchMark	Misprediction rate (%)
164.gzip	10.411
175.vpr	8.769
176.gcc	5.338
181.mcf	11.479
186.crafty	2.704
197.parser	6.336
201.compress	6.432
202.jess	0.655
205.raytrace	0.896
209.db	2.848
213.javac	1.536
222.mpegaudio	1.521
227.mtrt	0.952
228.jack	0.953
252.eon	0.587
253.perlbmk	0.750
254.gap	2.271
255.vortex	0.245
256.bzip2	0.044
300.twolf	13.231
average	3.897

contain a weight table access and a selector. A 4-level CSA follows the selector in stage 1, whereas a 1-level CSA and a carry propagation adder (CPA) follow in stage 0. The latency in accessing LHT in stage 0 is longer than that in the accessing weight table in stage 1. The latency of a 1-level CSA followed by a CPA is equivalent to that of 4-level CSA in this case of addition, where the CPA sums 12-bit numbers. Therefore, the critical path of the customized version is located in stage 0.

The critical path in stage 0 passes from the input of LHT to the prediction output through LHT, a 4-to-1 selector, a CSA, and a CPA whose latencies are denoted by T_{LHT}, T_{sel}, T_{CSA}, and T_{CPA}, respectively. They are estimated to be $T_{\mathrm{LHT}} = 17$, $T_{\mathrm{sel}} = 4$, $T_{\mathrm{CSA}} = 4$, and $T_{\mathrm{CPA}} = 12$ in units of 2-NAND gate latency. In the estimation we used the known expression $p + \log p + 1$ for the latency of a table with 2^p entries [10]. Consequently, the latency of the customized version is estimated to be 37 which is a little larger than 27, the estimated latency of the Path-Based Neural Predictor, all in units of 2-NAND gate latency.

As can be seen in Figure 11, the structure of our predictor is as simple as that of Path-Based Neural Predictor. So it is reasonable to consider that the complexity, area, and power assumption of our predictor are similar to those of the implementable Path-Based Neural Predictor.

Table 3: Storage requirements of our predictor.

Component	Size (bit)
Weight Tables	7bits $\times$ 1024 $\times$ 31 = 222208
Bias Weight Table	7bits $\times$ 1024 $\times$ 4 = 28672
LHT	2bits $\times$ 4096 = 8192
GHR	31
Weight Table Index	10bits $\times$ 32 $\times$ 2 = 640
LHT Index	12
MA	10bits $\times$ 31 $\times$ 2 = 620
Pipeline Registers	7bits + 9bits $\times$ 7 = 70
Total	260445

The storage requirements of our predictor are calculated as shown in Table 3. The hardware budget of our predictor is 260445 bits in total, which is less than that 32KB + 256bit = 262400 bits that can be used in the Realistic Track of the competition.

7. Conclusion

A novel pipelined perceptron branch predictor was presented. Our scheme uses local history as part of the index for weight tables. Another feature of our predictor is that it uses path history effectively: weight tables are accessed using the information that is available at the time and that is most probably related to the branch whose outcome is to be predicted. A version of our proposed predictor, which was shown to be reasonably implementable, has significantly better prediction accuracy than current predictors.

Acknowledgements

We would like to thank professor Daniel A. Jimenez for his valuable comments to improve the quality of the paper. We also would like to thank Dr. Kenji Kise of the Tokyo Institute of Technology for providing us with SimCore, an Alpha processor simulator. This research was supported in part by JSPS Grant-in-Aid for Scientific Research (C)(2) 18500048.

References

[1] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01)*, 2001.

[2] D. A. Jimenez, "Fast path-based neural branch prediction," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, 2003.

[3] D. A. Jimenez, "Piecewise linear branch prediction," in *Proceedings of the 32nd International Synposium on Computer Architecture (ISCA'05)*, 2005.

[4] Y. Ishii and K. Hiraki, "Path trace branch prediction," *IPSJ Trans. Advanced Computing Systems.*, vol. 47, no. SIG 3(ACS 13), pp. 58–72, 2006.

[5] S. McFarling, "Combining branch predictions," in *WRL Technical Note TN-36*, 1993.

[6] D. A. Jimenez, "Idealized piecewise linear branch prediction," in *in the First JILP Championship Branch Prediction Competition (CPB-1)*, 2004.

[7] K. Kise, T. Katagiri, H. Honda, and T. Yuba, "The simcore/alpha functional simulator," in *Workshop on Computer Architecture Education (WCAE-2004) held in conjunction with the ISCA-31*, 2004.

[8] Y. Ninomiya and K. Abe, "Path traced perceptron branch predictor using local history for weight selection," *IPSJ SIG Technical Reports*, vol. 2006, no. 88, pp. 31–36, 2006.

[9] Y. Ninomiya and K. Abe, "Path traced perceptron branch predictor using local history for weight selection," in *The 2nd JILP Championship Branch Prediction Competition (CBP-2) in conjunction with The 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39)*, 2006.

[10] J. E. Savage, *The Complexity of Computing.* Krieger Publishing Co., 1987.