NOLTA, IEICE

Paper

An energy-efficient dynamic branch predictor with a two-clock-cycle naïve Bayes classifier for pipelined RISC microprocessors

Itaru Hida [1 a)], *Shinya Takamaeda-Yamazaki* [1], *Masayuki Ikebe* [1], *Masato Motomura* [1], *and Tetsuya Asai* [1]

[1] *Graduate School of Information Science and Technology, Hokkaido University Kita 14, Nishi 9, Kita-ku, Sapporo, Hokkaido, Japan*

[a)] *hida@lalsie.ist.hokudai.ac.jp*

Received November 10, 2016; Revised February 8, 2017; Published July 1, 2017

Abstract: In this paper, we propose a Bayesian branch-prediction circuit, consisting of an instruction-feature extractor and a naïve Bayes classifier (NBC), as a machine learning approach for branch prediction. A branch predictor predicts the outcome of a branch instruction by analyzing the pattern of the previous branch outcome. In other words, branch prediction can be viewed as a type of pattern recognition problem, and such problems are often solved using neural networks. A perceptron branch predictor has already been proposed as one example of a neural branch prediction architecture, which predicts the next branch outcome by using past branch history to form feature vectors. The proposed circuit is constructed by replacing the arithmetic unit (neurons) in conventional neural branch predictors with an NBC. By introducing an approximate Bayesian computation and its parallel architectures, the NBC circuit completes branch prediction within two clock cycles per instruction. This constitutes a suitable replacement for conventional branch predictors in modern pipelined reduced instruction set computing microprocessors.

Key Words: dynamic branch prediction, supervised machine learning, naïve Bayes classifier, energy-efficient microprocessor, low-power architecture, CMOS digital circuit

1. Introduction

The artificial neuron was first proposed around half a century ago in [1–3]. Today, neural networks continue to result in significant improvements in the field of pattern recognition. Pattern recognition is used to identify the category of some given data that is useful for humans, such as images, sounds, texts, and statistical data, by extracting and learning its features [4–6]. However, useful information for computers could also become a target for pattern recognition. Indeed, general microprocessors are equipped with dynamic branch predictors that predict the outcome of a branch instruction by

Nonlinear Theory and Its Applications, IEICE, vol. 8, no. 3, pp. 235–245 DOI: 10.1587/nolta.8.235

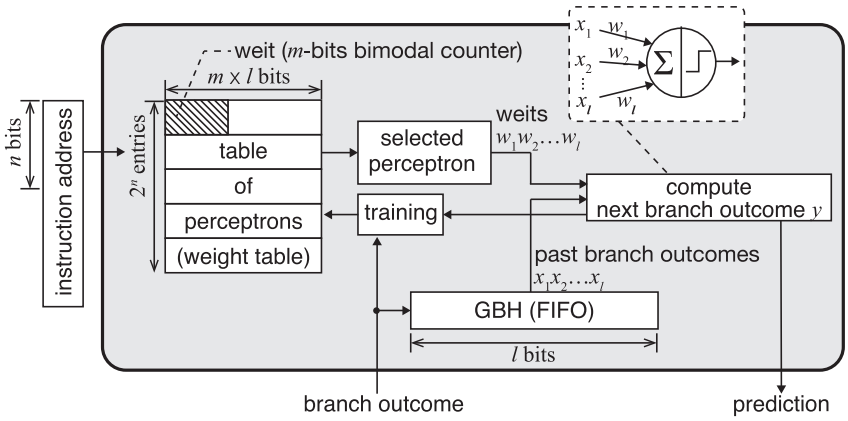


Fig. 1. Perceptron branch predictor [7].

analyzing the outcome pattern of a previous branch. Recent studies have proposed the adoption of a neural network approach to predict branches, and have proved that a neural branch predictor can achieve a significantly lower misprediction rate than conventional branch predictors [7].

Branch prediction represents one of the most important technologies required to achieve a highly effective performance in modern pipelined microprocessors. Its precision directly affects both instruction throughput and power consumption, because incorrect predictions can result in a pipeline stall (flush) and energy-inefficient instruction refetching from the (cache) memory. Many advanced prediction methods have been proposed and implemented with the aim of achieving high-precision branch prediction [9–11]. Among these, neural branch prediction, which is based on a machine learning approach, demonstrates a high prediction accuracy [8].

The original neural branch predictor consists of a linear classifier, a first-in first-out (FIFO) global branch history (GBH) of length l, and a weight table, as shown in Fig. 1, where the variable m represents the number of bits of the weight counter, x represents past branch outcomes, and y represents the predicted next branch outcome. The FIFO GBH acts as an instruction-feature extractor. In addition, the linear classifier is trained on the basis of a simple perceptron rule, with l different weight sets targeted by the addresses of the branch instruction. Because learning is always performed, weight divergence or learning may be observed during online learning. This significantly degrades the precision of the branch prediction. To solve this problem, we focus on the use of a naïve Bayes classifier (NBC) [12], and replace the linear classifier in Fig. 1 with NBC units.

NBCs have been applied as mathematical formalisms within the field of machine learning in various domains, ranging from medical informatics to e-mail filtering. Several NBC circuits have been proposed, such as in [13, 14]). In general, the computational cost of an NBC is considerably high, on account of the additive and multiplicative arithmetical operations involved. Indeed, in [13], a minimum of five clock cycles were required for NBC computation, whereas in [14] 10 clock cycles were required. In standard pipelined microprocessors, for example those consisting of five stages indexed by instruction fetch (IF), instruction decode (ID), execution (EX), memory access (MA), and write back (WB), branch prediction following IF must be completed before the conclusion of the EX stage. Hence, when employing NBC the computation for branch prediction must be finished within two clock cycles. Therefore, in this paper we propose a short-latency naïve Bayes classifier. By integrating NBC circuits in an open-source 32-bit microprocessor (LatticeMico32 [15]), we demonstrate the performance in terms of both precision and energy efficiency for the static, perceptron, and proposed NBC branch predictors.

2. Architecture

2.1 Naïve Bayes classifier

Naïve Bayes is the simplest type of Bayesian network. It is a probabilistic graphical model, representing conditional dependencies of random variables through a directed acyclic graph. A naïve Bayes network consists of multiple child nodes and a single parent node. Each child represents an evidence

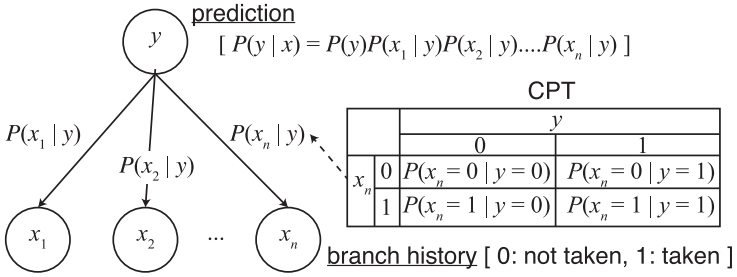


Fig. 2. NBC for branch prediction: feature variables x_i represent previous branch outcomes, and the class variable y represents the predicted next branch outcome.

variable x_i, for which the parent is the class value. A child and its parent are attached through a conditional probability table (CPT). We can predict a class value by calculating the posterior probability $P(y = c \mid x_1, x_2, \ldots, x_n)$, using CPTs and the prior probability $P(y)$, as follows:

$$P(y = c \mid X) = \frac{P(x_1 \mid y = c)P(x_2 \mid y = c) \ldots P(x_n \mid y = c)P(y = c)}{P(x_1, x_2, \ldots, x_n)} \tag{1}$$

In terms of branch prediction, the evidence variables $X = \{x_1, x_2, \ldots, x_n\}$ represent previous branch outcomes, and the class value y represents the predicted next branch outcome (see Fig. 2). Each x_i and y take a value of 0 (not taken) or 1 (taken). Thus, all variables in the network are Boolean. Using these variables, we calculate the posterior probabilities $P(y = 0 \mid X)$ and $P(y = 1 \mid X)$, and predict the next branch outcome by comparing these probabilities.

2.2 Hardware architecture of NBC

Figure 3 illustrates the hardware architecture of an NBC. We implement each likelihood $P(x_i \mid y)$ in the CPT and prior probability $P(y)$ by using up/down saturating bimodal counters, given that each x_i is Boolean and the following relationship holds:

$$\begin{cases} P(y = 0) = 1 - P(y = 1) \\ P(x_i = 0 \mid y = c) = 1 - P(x_i = 1 \mid y = c) \end{cases} \tag{2}$$

As shown in Fig. 3, there are two counters $p(y = 0)$ and $p(y = 1)$ in the $P(y)$ table. These counters are updated when a branch instruction is executed, as follows:

if $t = 0$ **then**
  $p(y = 0) \leftarrow p(y = 0) + 1$
  $p(y = 1) \leftarrow p(y = 1) - 1$

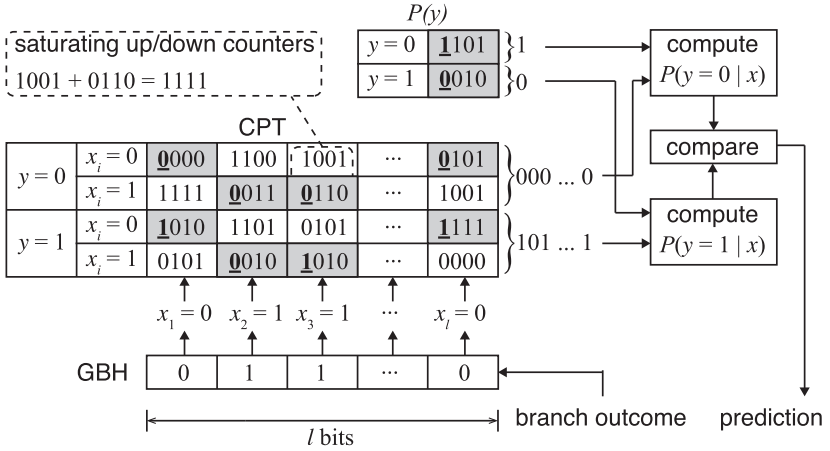


Fig. 3. Hardware architecture for calculating the posterior probabilities by using a CPT. Probability estimates are simplified for hardware implementation.

end if

where t is the branch outcome. The same rule applies when $t = 1$.

The GBH registers and the function are implemented as shift registers and an instruction-feature extractor, respectively. For each historical outcome x_i, there are four counters $p(x_i, y)$, where x_i and y are either 0 or 1. These counters are updated in a similar manner to the $P(y)$ table, as follows:

if $t = 0$ **then**
  if $x_i = 0$ **then**
    $p(x_i = 0, y = 0) \leftarrow p(x_i = 0, y = 0) + 1$
    $p(x_i = 1, y = 0) \leftarrow p(x_i = 1, y = 0) - 1$
  end if
end if

Using these counters, we calculate $P(y = 0 \mid X)$ and $P(y = 1 \mid X)$. By substituting these probabilities into Eq. (1), the fractions in the formulae can be eliminated, because we only need to compare them to predict the next branch outcome. Therefore, we obtain

$$P(y \mid X) \propto P(y)P(x_1 \mid y)P(x_2 \mid y) \ldots P(x_n \mid y) \tag{3}$$

Furthermore, to replace multipliers with adders, we compute the logarithm of both sides of Eq. (3), as follows:

$$\log P(y \mid X) \propto \log P(y) + \log P(x_1 \mid y) \ldots + \log P(x_n \mid y) \tag{4}$$

Figure 3 shows the prediction of the next branch outcome. Appropriate counters are selected between $x_i = 0$ and $x_i = 1$ from the $y = 0$ and $y = 1$ rows for each i, based on the values in the GBH.

As shown in Fig. 4, the most significant bits (MSBs) in the selected counters are added, because it is possible to replace the additions of the logarithms in Eq. (4) with summations of the MSBs in the selected counters [16]. We implemented this addition function by using lookup tables (LUTs) and several small adders to reduce the latency and circuit area. Because each MSB loaded from the CPT or $P(y)$ table represents a single bit, the addition of these bits can be replaced by counting the number of "1" bits. Hence, the loaded MSBs are first segmented into small groups, and the number of bits that are set to 1 in each group are counted using the LUTs. Next, the outputs from the LUTs are added using some small adders. Moreover, pipeline registers are inserted between the adders, because the path through these chained adders can be critical.

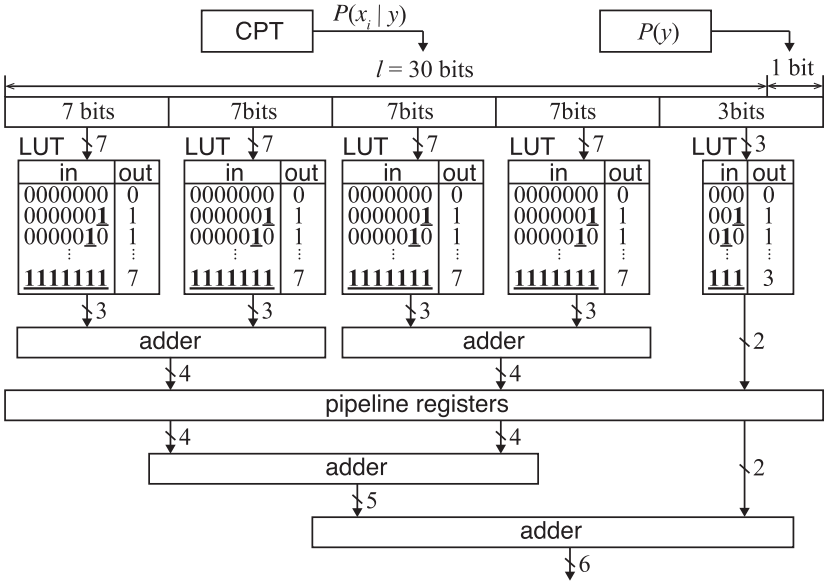


Fig. 4. Computation of posterior probability $P(y = c \mid x_1, x_2, \ldots, x_n)$ using LUTs and adders.

2.3 Implementation of naïve Bayes branch predictor in a microprocessor

To determine the pipeline depth of the proposed naïve Bayes branch predictor (NBBP), we examined the pipeline stages of LatticeMico32 [15], which is the base processor used to implement the NBBP. LatticeMico32 has six pipeline stages, as shown in Fig. 5, and branches are predicted in Stage D. This indicates that only one clock cycle elapses between a prediction in Stage D and the execution in Stage X. Considering the latency of the NBC architecture, it is impractical for the NBBP to predict the next branch outcome in one clock cycle. Accordingly, we applied the branch decoder and predictor to Stage F instead of Stage D. Thus, we have two clock cycles between the prediction and execution, as illustrated in Fig. 6. We inserted pipeline registers into the adders, and divided the adders into two stages, as shown in Fig. 4.

Figure 7 presents the schematic of the proposed NBBP. The branch instruction address is hashed to select a CPT and a $P(y)$ table, and the GBH functions as an instruction-feature extractor. The

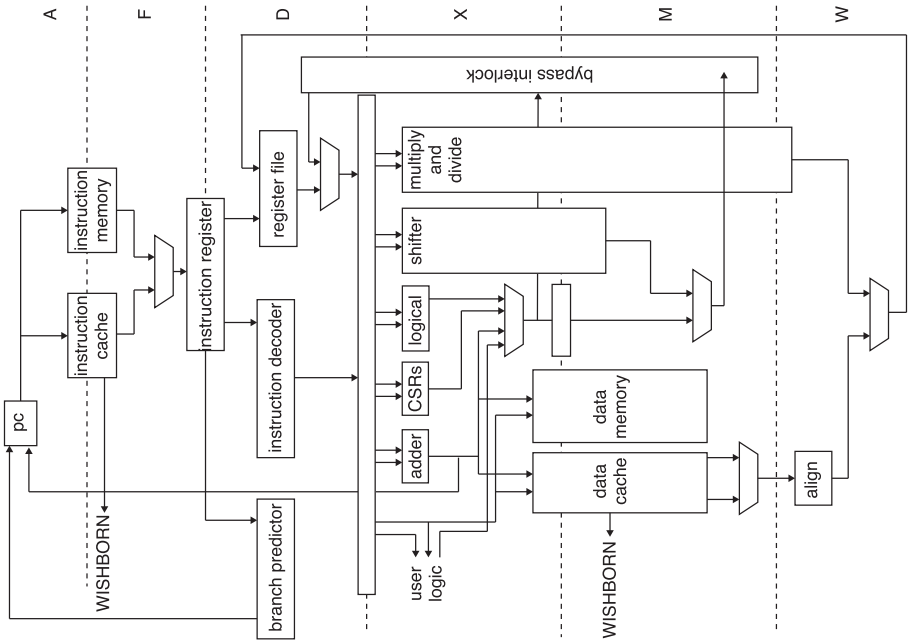


Fig. 5. Pipeline stages of LatticeMico32 [15].

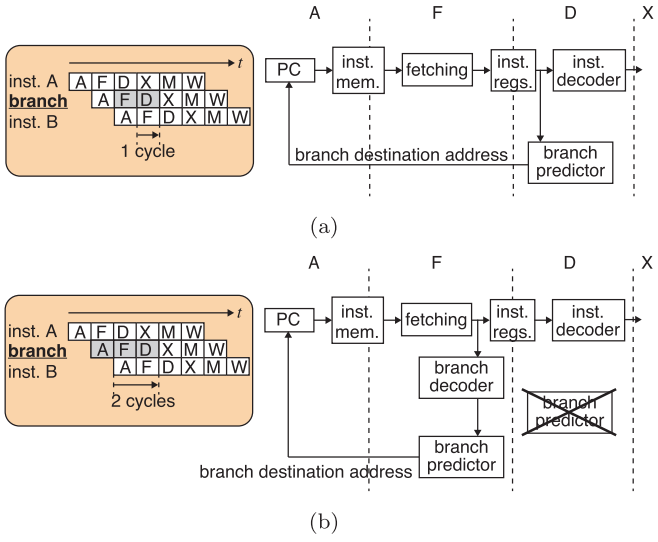


Fig. 6. Relocation of the branch predictor from Stage D to F in LatticeMico32. (a) Branch prediction is operated in Stage D and executed in Stage X. (b) We relocated the prediction to Stage F to achieve execution within two clock cycles.

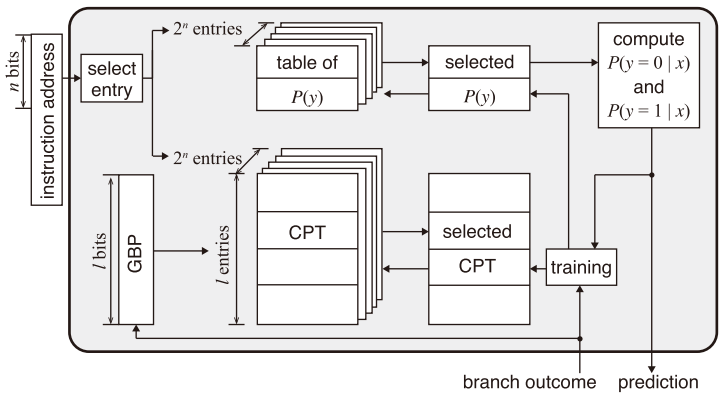


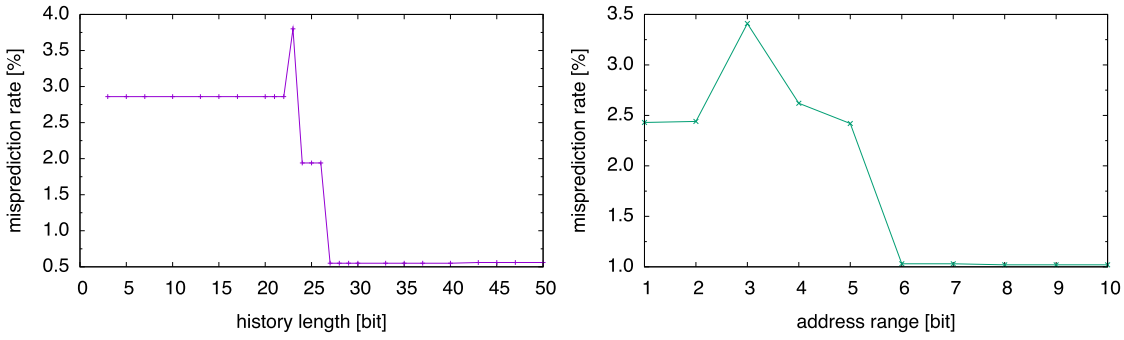
Fig. 7. NBBP Architecture.

posterior probabilities $P(y = 0 \mid X)$ and $P(y = 1 \mid X)$ are computed using the selected $P(y)$ and CPT, and the next branch outcome is predicted by comparing them. When the branch instruction is executed, the GBH, $P(y)$, and the CPT are updated according to the branch outcome. The prediction accuracy and hardware cost are mainly dependent on the sizes of the CPTs, for example the length of the GBH (l bits), range of the branch addresses used for hashing, and sizes of the bimodal counters in the CPTs. Next, we investigated the dependencies among these parameters and the prediction accuracy through simulations.

3. Simulation results

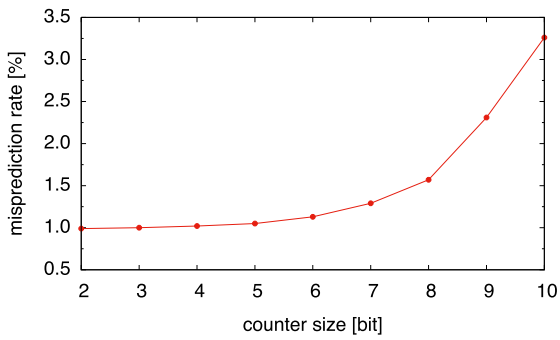
3.1 Misprediction rate

We used the MiBench benchmarks [17] to assess the accuracy of the proposed NBBP method implemented in LatticeMico32. First, we investigated the effects of the history length, address range, and counter size on the accuracy. It is clear from previous experiments that a greater history length and address range imply a higher accuracy, as shown in Fig. 8. In contrast, the accuracy decreases



(a) address range = 8, counter size = 4

(b) history length = 30, counter size = 4



(c) history length = 30, address range = 8

Fig. 8. Simulation results for the misprediction rate. (a) History length vs. misprediction. (b) Address range vs. misprediction. (c) Counter size vs. misprediction.

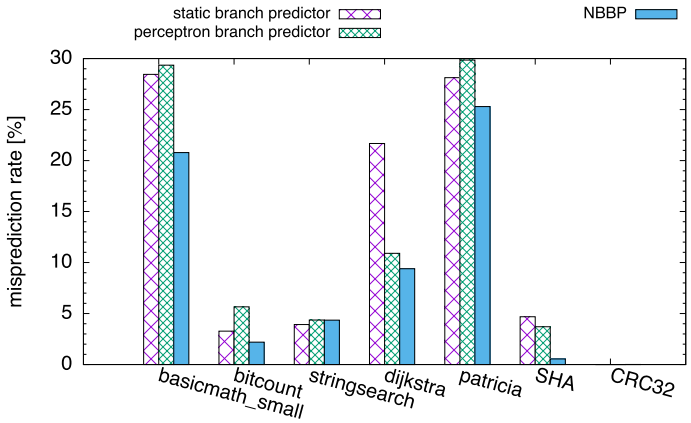


Fig. 9. Performance comparison of the NBBP and conventional branch predictors.

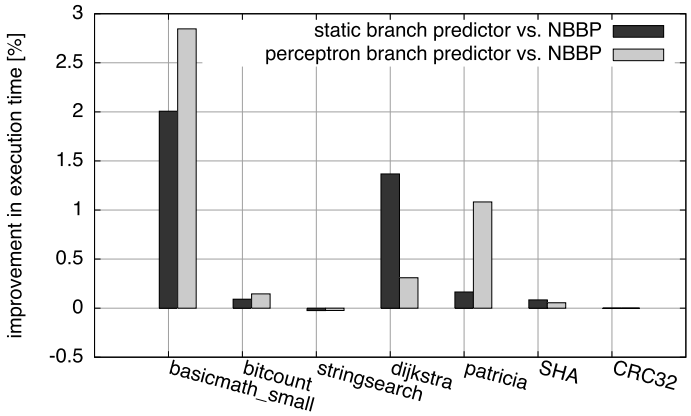


Fig. 10. Improvement in execution time of each benchmark.

as counter size increases, because a large counter requires a long time to converge. Based on these results, we set the history length to 30 bits, address range to eight bits, and size of each bimodal counter to four bits.

Then, we compared the accuracy of our proposed NBBP method with the static and perceptron branch predictors, by using seven benchmarks available in MiBench. The static branch predictor is the standard branch predictor implemented in LattieMico32. It always predicted that forward-pointing conditional branches would not be taken, and that instead backward-pointing conditional branches and unconditional branches would be taken.

The results of the simulation showed that NBBP is the most accurate method for five of the applications, as shown in Fig. 9. Naïve Bayes was able to learn linearly inseparable functions, unlike simple perceptrons. Thus, NBBP outperformed the perceptron branch predictor. For the other two applications, the static branch predictor yielded the highest accuracy. We assume that these two applications have a very simple structure or include few iterations.

Figure 10 shows the improvement in the execution time. The execution time was only slightly reduced, even with the proposed high accuracy NBBP method. The degree of the speed-up effect depends on the scale of the program and the number of stages of the processor's pipeline. Hence, using larger benchmark programs or a more complex microprocessor would have a significant effect on both the execution speed and the prediction accuracy.

3.2 Power consumption

We estimated the power consumption of LatticeMico32 equipped with the static branch predictor or NBBP by using its toggle rate and MA frequency, obtained through simulations. For the estimation conditions, we set the design technology to 0.18 μm, the core voltage to 1.8 V, and the clock frequency to 100 MHz, as shown in Table I, and employed the basicmath_small benchmark for the estimation.

Table I. Synthesis conditions for power estimation.

	Specification
Technology	UMC 0.18 μm CMOS
Core area	3.0 mm $\times$ 3.0 mm (CPT accounts for 70 %)
Supply voltage	1.8 V
Clock frequency	100 MHz
Size of CPT	122 kbits
Size of $P(y)$ table	2 kbits
Size of instruction memory	800 kByte
Size of data memory	800 kByte

To prepare for the simulation, we first synthesized the verilog source files of LatticeMico32 using a Synopsys design compiler. The results of the synthesis showed that registers of the CPT and the $P(y)$ table accounted for a large portion of the core area. We considered that these registers would lower the energy efficiency, and hence assumed that the CPT and the $P(y)$ table should not be implemented as registers, but rather as embedded SRAM. We then estimated their power consumption by using the given data from [18].

As previously mentioned, the NBBP method is more accurate than the static branch predictor. A high prediction accuracy reduces the frequency of communication operations, such as IF, between the microprocessor and memory, thus enhancing the energy efficiency. To consider this advantage of NBBP, we estimated the power consumed by accessing memory based on the prepared data [18].

Figure 11 illustrates the simulation model, and Table II presents the estimation results for one iteration of the benchmark. P_{sum} and $P_{\mathrm{ave.}}$ represent the power consumed during the execution and the average power consumption, respectively.

"Logic" represents the power consumption of the LatticeMico32 circuit. When NBBP was implemented, more power was consumed than for the static branch predictor. As shown in Table III, the number of gates of the processor in which each branch predictor was implemented was 30.6 K for the static branch predictor, 44.8 K for the perceptron branch predictor, and 51.4 K for NBBP. This gate increase led to a decrease in the logic power efficiency. However, because LatticeMico32 only loses

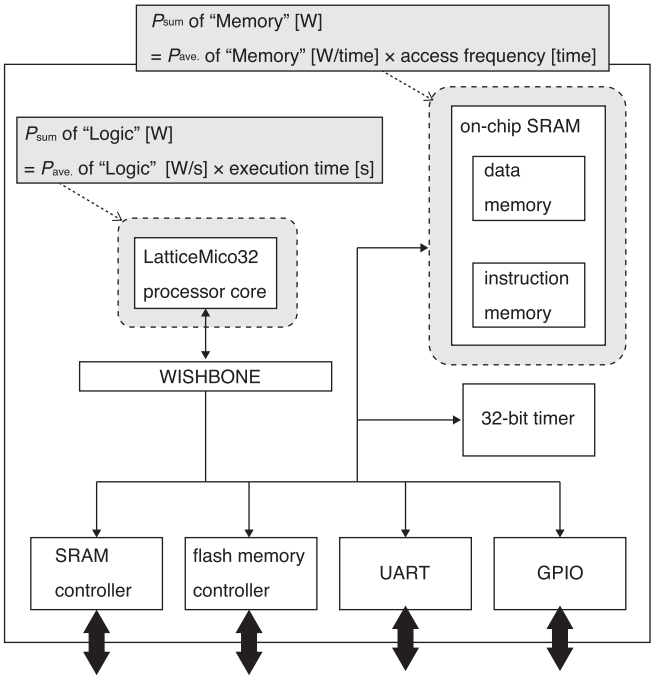


Fig. 11. Block diagram of the LatticeMico32 embedded system. We estimated the power consumption of the processor core as well as the data and instruction memories.

Table II. Simulation and estimation results when running basicmath_small benchmark in one iteration ($n = 1$).

	Logic			Memory			Total power consumption
	$P_{\text{l\_ave.}}$	Exe. time	$P_{\text{l\_sum}}$	$P_{\text{m\_ave.}}$	Access freq.	$P_{\text{m\_sum}}$	
Static branch predictor	12.7 mW	3.03 ms	0.039 mW	70 nW	295 ktimes	20.7 mW	20.73 mW
NBBP	41.9 mW	2.97 ms	0.124 mW	70 nW	289 ktimes	20.3 mW	20.42 mW

Table III. Gate count and circuit area of LatticeMico32.

	Gate count [K]	Circuit area [mm^2]
w/ static branch predictor	30.6	0.287
w/ perceptron branch predictor	44.8	0.382
w/ NBBP	51.4	0.482

three clock cycles when failing to predict a branch, the difference in the execution time between the static branch prediction and NBBP was 0.06 ms. Therefore, although the number of gates increased, the overhead of the logic power was only 0.09 mW.

On the other hand, the high accuracy of the branch prediction using NBBP led to a reduction in the memory access frequency, thus reducing the power consumption for memory access. In Table II, "Memory" represents the estimated power consumption resulting from memory access. The processor accessed the memory approximately 300,000 times during execution of 3 ms. The difference in the number of times memory was accessed between the static branch prediction and NBBP was 6000, and the power of the memory access was reduced by 0.4 mW.

As a result, the reduction of the "Memory" power exceeded the increase in the "logic" power consumption. Hence, the power efficiency of the overall system was improved.

Because this power reduction effect becomes more significant as the program becomes larger, we determined that the processor using NBBP is more efficient than the other in terms of the total power consumption.

Using the simulation and estimation results, we calculated the logic power consumption when iterating the benchmark as follows:

$$P_{\text{l\_sum}}\ [\text{W}] = P_{\text{l\_ave.}}\ [\text{W/s}] \times \text{execution time [s]} \times n$$

where n is the number of iterations. Similarly, the memory power consumption is estimated by

$$P_{\text{m\_sum}}\ [\text{W}] = P_{\text{m\_ave.}}\ [\text{W/time}] \times \text{access freq. [time]} \times n$$

The results we obtained for these calculations are shown in Fig. 12. We determined that the difference

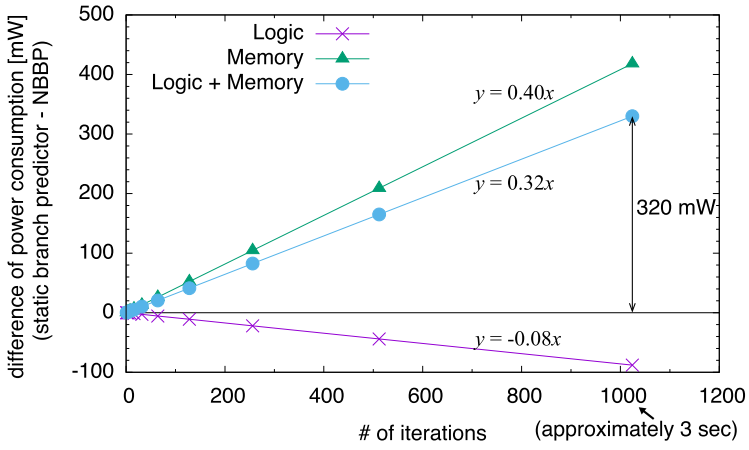


Fig. 12. Dependence of the difference in power consumption on the number of benchmark iterations.

between the power consumptions of the processors using NBBP and the static branch predictor satisfies the relation

$$\text{difference of power consumption [mW]} = 0.32n$$

, and the processor using NBBP consumed 320 mW less power than the processor using the static branch predictor.

4. Conclusion

In this paper, we have proposed a highly accurate dynamic branch predictor that employs an NBC. The proposed NBBP method predicts the next branch outcome within two clock cycles, by simplifying probability calculations using Bayes' theorem. In our simulation, an RISC processor equipped with NBBP yields a higher accuracy than conventional branch predictors. The estimation results demonstrated that the proposed architecture is more energy efficient than static branch prediction. The high prediction accuracy of this architecture reduces pipeline stalling and instruction refetching. Thus, NBBP is a valid method for improving the energy efficiency of modern pipelined processors.

Acknowledgments

This study was supported by the JSPS Grants-in-Aid for JSPS Fellows, and a Grant-in-Aid for Scientific Research on Innovative Areas [2511001503] from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan.

References

[1] W.S. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[2] D. Hebb, *The Organization of Behavior*, Wiley, 1949.

[3] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[4] P.Y. Simard, D. Steinkraus, and J. Platt, "Best practice for convolutional neural networks applied to visual document analysis," in *Proc. ICDAR 2003*, p. 958, 2003.

[5] D.C. Ciresan, U. Meier, J. Masci, L.M. Gambardella, and J. Schimidhuber, "Frexible, high performance convolutional neural networks for image classification," in *Proc. IJCAI*, 2011.

[6] G.E. Hinton, L. Deng, D. Yu, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, and V. Vanhoucke, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, 2012.

[7] D.A. Jimeńez and C. Lin, "Dynamic branch prediction with perceptrons," in *Proc. 7th Int. Symp. High Performance Computer Architecture*, pp. 197–206, 2001.

[8] D.A. Jimeńez, "Oh-snap: Optimized hybrid scaled neural analog predictor," in *Proc. 3rd Championship on Branch Prediction*, 2011.

[9] T.Y. Yeh and Y.N. Patt, "Two-level adaptive training branch prediction," in *Proc. 24th Annual International Symposium on Microarchitecture*, pp. 51–61, 1991.

[10] M. Evers, P.Y. Chang, and Y.N. Patt, "Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches," in *Proc. 23th International Symposium on Computer Architecture*, pp. 3–11, 1996.

[11] C.C. Lee, I.C.K. Chen, and T.N. Mudge, "The bi-mode branch predictor," in *Proc. 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 4–13, 1997.

[12] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, (2nd ed.), Prentice Hall, 2003.

[13] M.N. Marsono, M. Watheq El-Kharashi, and F. Gebali, "Binary LNS-based naïve Bayes inference engine for spam control: Noise analysis and FPGA implementation," *IET Comput. Digit. Tech.*, vol. 2, no. 1, pp. 56–62, 2008.

[14] D. Zhijie, W. Yong, and T. Xiaoling, "Method of network traffic classification using naïve Bayes based on FPGA," in *Proc. 2011 IEEE Int. Conf. Intelligent Computing and Integrated Systems*, pp. 1–3, 2013.

[15] http://www.latticesemi.com/mico32

[16] J. Singer, G. Brown, and I. Watson, "Branch prediction with Bayesian networks," in *Proc. Workshop on Statistical and Machine Learning Approaches Applied to Architectures and Compilation*, pp. 96–112, 2007.

[17] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. 4th Workshop on Workload Characterization*, pp. 3–14, 2001.

[18] http://www.csd.uoc.gr/~hy534/03a/s31_ram_bl.htm