Introducing Entropies for Representing Program Behavior and Branch Predictor Performance

Takashi Yokota Kanemitsu Ootsu Takanobu Baba
Utsunomiya University
7–2–1 Yoto, Utsunomiya, 321-8585 Japan
{yokota, kim, baba}@is.utsunomiya-u.ac.jp

ABSTRACT

Predictors are inherent components of state-of-the-art microprocessors. Branch predictors are discussed actively from diverse perspectives. Performance of a branch predictor largely depends on the dynamic behavior of the executing program. Nevertheless, we have no effective metrics to represent the nature of program behavior quantitatively. In this paper, we introduce an information entropy idea to represent program behavior and branch predictor performance. Through simple application of Shannon's information entropy, we introduce new entropy, Branch History Entropy, which quantitatively represents the regularity level of program behavior. We show that the entropy also represents an index of prediction performance that is independent of prediction mechanisms. We further discuss branch predictor performance from a stereoscopic view of their typical organization. We propose two entropies: Table Reference Entropy and Table Entry Entropy. The former represents an unbalanced level of references of table entries. The latter offers the maximum expectation in prediction performance. We evaluated the proposed three entropies and prediction performance in various situations. Artificially generated branch patterns, as preliminary experiments, show an overview of the entropies and prediction performance. Subsequently, we present a comparison to the 2nd Championship Branch Predictor competition results and show the high potential of the proposed entropy. Finally, we present an actual view of our entropies and prediction performance as application results to SPEC CPU2000 benchmarks.

Categories and Subject Descriptors

C.1 [**Processor Architectures**]: General

General Terms

Architecture

ExpCS 13–14 June 2007, San Diego, CA

Keywords

branch predictors, information entropy, program behavior, prediction performance

1. INTRODUCTION

A predictor serves an important role in state-of-the-art microprocessors. A branch predictor is an inherent component of microprocessor architecture. Methodologies and their effects in branch predictors are discussed actively from many perspectives. Furthermore, some competitive discussions were made for the Championship Branch Predictor (CBP, [19, 20]), seeking realistic and idealistic branch prediction methodologies.

Most branch predictors have a common structure in logic. They accumulate meaningful information from past events that occurred in the processor microarchitecture. They then select the most likely event to occur in the near future, by consulting the accumulated information. For example, some simple branch predictors use only the latest result. They predict that the same event will occur again the next time. As another example, a perceptron, one neural network method, is applied to a branch predictor. This complex predictor also stores past events by *learning* and predicts the most likely event.

Because most predictors use past events to predict coming events, strong regularity is required. For example, a predictor seeks the latest sequence of events in its event storage. It predicts the next event with strong confidence when it finds frequent occurrence of the same sequence in the past. However, when the system has weak regularity, the prediction is not always a hit because of the low repeatability of the event sequence. Consequently, most predictors rely strongly on the regularity level in the event sequence. On the other hand, average metrics only partially rely on the regularity level.

In this paper, we specifically examine branch predictors to clarify our discussion. The performance of branch predictors is reliant on their performance on the regularity level in a sequence of conditional branch results; average metrics, such as branch probability, are not addressed herein. Regularity comes from program execution: program behavior. We mean that a branch predictor achieves good performance if the program behavior has sufficiently high regularity.

Furthermore, we claim that there should be a strong correlation between predictor performance and the program's regularity level. Moreover, if the regularity level is represented numerically as an index, the index also represents prediction performance. However, we so far have no effec-

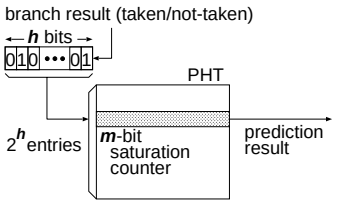


Figure 1: Two-level branch predictor organization.

tive index that represents, quantitatively, the program's regularity levels.

A major objective of this paper is to offer theoretical and quantitative indices of program behavior and prediction performance. We introduce *information entropy* to solve the problem. Information science has a firm theoretical foundation that originated from Shannon. A high level of regularity shows a low entropy value and high prediction performance is expected.

This paper presents a stereoscopic view of program behaviors and branch predictors using two aspects of entropies: essential information of program behavior, and information involved in the logical organization of branch predictors.

The remainder of this paper is organized as follows. We first introduce regularity in program behavior and prediction performance in Section 2. Subsequently, we introduce a new entropy measure *Branch History Entropy* in Section 3. Section 4 presents discussion of information involved in predictor organization. Section 5 shows our evaluation results. Section 6 describes related work and Section 7 concludes this paper.

2. REGULAR BEHAVIORS IN PROGRAM EXECUTION AND PREDICTORS

We know that branch predictors perform well if they have sufficient memory for storing past event information. However, it is true that twice the quantity of memory does not always achieve twice the predicted performance. As one example, the prediction miss ratio will not be reduced to half when memory is doubled. Actually, in many cases, prediction performance saturates at some level; worse yet, overly large amounts of memory sometimes cause undesired results.

2.1 Experiences in path prediction

We proposed the 'two-path limited speculation' method for efficient execution in multi-threaded processors[26, 25]. The method specifically emphasizes a certain loop in a program and predicts the execution path of the next iteration of the loop. Here we simply call it a *path*.

The method selects the two most frequently executed paths that are called number-1 and number-2 (#1 and #2) paths. The method predicts the next path of the two (#1 or #2 path), and speculatively executes the predicted path. The method uses a simple path predictor that is based on the well-known two-level branch predictor[22], as shown in **Fig. 1**.

The predictor estimates whether the #1 path is executed at the next time or not. At every loop iteration, the path execution history is stored into the history register. One '1' bit is inserted at the LSB of the history register after shifting left one bit if #1 path is executed. The history register points to an entry in a pattern history table (PHT). The PHT has 2^h entries where the h-bit history register is used. Each PHT entry consists of a small saturation counter, of two bits; it counts the number of executions of the #1 path. The counter is incremented if the #1 path is executed. Otherwise, the counter is decremented. Consequently, when the PHT entry that is indicated by the history register shows a large value, the #1 path is predicted.

Many SPEC CINT95[18] programs show the following interesting phenomena.

1. Only very few paths, one or two, are executed frequently; the remaining paths are scarcely executed.
2. The prediction hit ratio increases when the history length is greater than 10 (bits).
3. A time-sequence plot of the prediction hit ratio shows an 'execution pattern.'

Table 1 shows the occurrence ratios of #1 and #2 paths. **Fig. 2** shows prediction hit ratios for various history lengths, and **Fig. 3** shows time sequence plots of the prediction hit ratio, measured in every 10,000 clock cycles. `compress` is the compress() function in 129.compress, `forward_DCT` is the forward_DCT() function in 132.ijpeg, `killtime` is the killtime() function in 124.m88ksim, and `sweep` is the sweep() function in 130.li in SPEC CINT95 benchmarks.

Here we present some simple questions. Why does the prediction hit ratio increase only when the history length is greater than 10? Figure 3 shows that the prediction hit ratio increases from the beginning of program execution and that some programs have a certain *pattern*. What does it mean?

We have investigated the predictor behavior to answer the first question. **Table 2** shows the number of occurrences of history register values, sorted using occurrences. Program is `compress` and the 8-bit history is used. The total number of history occurrences is 5751. The table shows entries whose reference count is more than 50 and omits less-referenced entries. The number of entries that were referred at least once is only 90, whereas the PHT has 256 entries. The result clearly gives extremely unbalanced references, where occurrence probabilities of #1 and #2 paths are not extremely unbalanced, as shown in Table 1.

What does it mean? If the #1 path occurs randomly according to its corresponding frequency, as given in Table 1, frequently occurring history values should be widely distributed. However, Table 2 shows the opposite result. We have investigated the transition of history values. The history register contains the most recent h execution results of iterations. A history value $\{h_{h-1}, h_{h-2}, ..., h_1, h_0\}$ has two successive values $\{h_{h-2}, ..., h_1, h_0, x\}$, where $x = 0$ or 1. **Table 3** shows percentages of history values whose successive value (x) dominates more than 90%. For example, in the case of the `compress` program with a 12-bit history, 22.5% of history values transit to its peculiar descendant, i.e., 'extremely unbalanced' transition. The ratio of the extremely unbalanced transition increases when the history length is greater than 10 bits. These results correspond to those of hit ratio evaluation shown in Fig. 2.

Therefore, we have discerned these facts: unbalanced references in the predictor increase prediction performance; such imbalances stem from program behavior.

Table 1: Occurrence ratios of the top two paths in SPECint95 programs.

function	#1 path	#2 path
`compress`	54.5	22.4
`forward_DCT`	48.2	42.1
`killtime`	97.0	3.0
`sweep`	80.7	19.3

occurrence ratio [%]

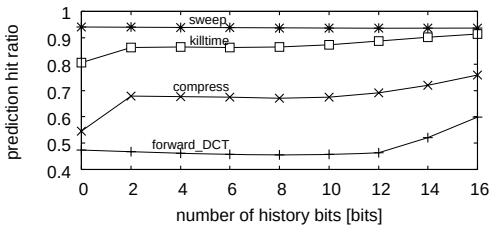


Figure 2: Prediction hit ratios for various history lengths.

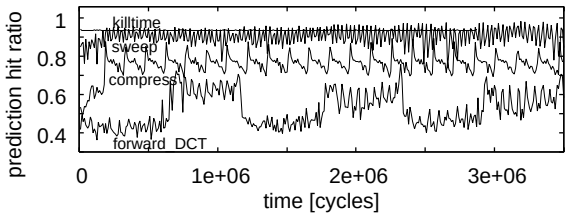


Figure 3: Time sequence of hit ratios of the path predictor.

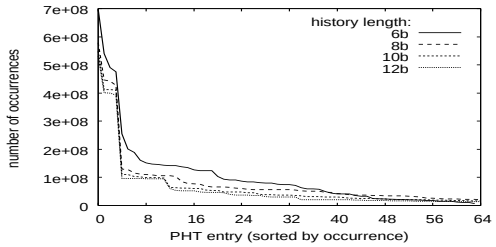
Table 2: Numbers of occurrences of history values in the `compress` program.

number of occurrences	history values
975	00000000
325	11101111
307	11110111
257	11011111
241	11111011
225	11111111
197	10111111
183	11111101
151	00100000
145	01111111 11111110
125	00000100 00001000 00010000
77	01000000
75	00000001 01000001 10000010
74	00000010 10000000 10111011 11011101
68	11011110
66	01111011 10111101
60	10111110
58	01111101
55	01110111
52	01111110
51	01001000 10010000

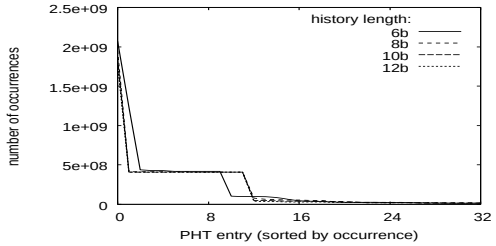
Table 3: Ratios of extremely unbalanced transitions in history values.

	history length			
function	4	8	12	16
`compress`	0.0	4.1	22.5	48.3
`forward_DCT`	0.0	0.0	1.3	10.3
`killtime`	95.9	95.9	95.9	95.9
`sweep`	64.4	60.0	65.2	76.0

unit: %



(a) 164.gzip



(b) 256.bzip2

Figure 4: Distribution of PHT references for various history lengths.

2.2 Unbalanced behavior in the branch predictor

The path predictor, introduced in the previous section, concerns only a single loop, whereas branch predictors predict for every branch in a program. For that reason, it is expected that PHT references are widely distributed because a variety of branch instructions are executed and the branch predictor predicts their results.

However, branch predictors present similar behavior to that of the path predictor, i.e., it is the opposite result from our expectation. **Fig. 4** shows the distribution of the number of references of PHT entries in some history length conditions. The numbers are sorted into descending order. Only the largest 64 and 32 entries are shown respectively in Figs. 4(a) and 4(b). The remaining small numbers are omitted from the graphs. Executed programs are `164.gzip` and `256.bzip2` in the SPEC CINT2000[17] benchmark and two-level branch predictors are used. In `164.gzip` program execution, about 59.4% of conditional branches are taken; the rest are not-taken.

According to the so-called *10–90 rule*, a small fraction (10%) of the program dominates the execution time (90%). Consequently, we infer that 10% of branch instructions are frequently executed and that the remainder are not so frequently executed. Not a few branch instructions are involved in the 10% portion. However, their behaviors of taken

and not-taken are quite regular and PHT references are extremely imbalanced.

Furthermore, Fig. 4 shows that the effective number of frequently referred PHT entries is almost independent of the history length. For that reason, the program has strong regularity in execution of branch instructions. The number of frequent entries increases if any branch instruction shows random behavior.

3. REPRESENTING PROGRAM BEHAVIORS

Section 2 shows a strong suggestion that regularity in program behavior causes an imbalanced situation in PHT references in a branch predictor in addition to good prediction performance[23, 24]. The resulting index should be useful for discussing branch predictor architecture if we can represent the regularity level of program execution quantitatively.

3.1 Lessons from Shannon

We first trace Shannon's information entropy[14].

Assume that we are discussing an entropy $H(S)$ of a Markovian information source S that produces a string of symbols. Instead of the entropy of S itself, we first discuss the augmented adjoint source of S. An n-th order augmented adjoint source of S, i.e., $\overline{S}^n$, generates n consecutive symbols. The entropy of the n-th order augmented adjoint source $H(\overline{S}^n)$ is given as the following equation:

$$H(\overline{S}^n) = -\sum_i p(S_i^n) \log_2 p(S_i^n), \quad (1)$$

where $p(S_i^n)$ represents the probability (i.e., occurrence ratio) of an individual symbol S_i^n that comprises consecutive n original symbols.

The n-th approximation of the entropy $H(S)$ of the targeted Markovian information source is given as the following equation when $(n+1)$-th order augmented adjoint entropy $H(\overline{S}^{n+1})$ is given.

$$H^n(S) = H(\overline{S}^{n+1}) - H(\overline{S}^n) \quad (2)$$

Therefore, the true value of the targeted entropy is given by limiting n to infinity:

$$H(S) = \lim_{n\to\infty} H^n(S). \quad (3)$$

The entropy $H(S)$ provides essential information of the next symbol. The entropy also presents the predictability of the forthcoming symbol.

3.2 Branch History Entropy

Discussions presented in the previous subsection lead us to an entropy concept of quantitative representation of program behaviors. Here, we discuss a simple application of Shannon's information entropy into program behaviors.

We can consider that a branch predictor inputs a string of branch results, i.e., taken or not-taken, from the instruction execution part of the processor and the predictor predicts the forthcoming branch result. For that reason, the processor is an information source that produces a string of 1-bit information of branch results (taken/not-taken). The information source has a Markovian property.

A sequence of n consecutive branch results is considered as a symbol of n-th order augmented adjoint information source. In addition, simple application of Eq. (1), n-th order augmented adjoint entropy is given. Furthermore, as in the preceding subsection, n-th order approximation of entropy is given as Eq. (2), and the true value of the information source is represented as Eq. (3).

We call the entropy *Branch History Entropy*, denoted as $H(B)$. We denote the n-th order augmented adjoint entropy as $H(\overline{B}^n)$ and the n-th order approximation of entropy as $H^n(B)$.

3.3 Discussion

Quantitatively, Branch History Entropy represents essential information involved in a sequence of branch results, i.e. program behavior. Because Shannon's information entropy shows the predictability of the subsequent symbol, the entropy value represents the predictability of the forthcoming branch result. The system has binary events (0 or 1). Therefore, the entropy value $H(B)$ is less than or equal to 1.0.

Information entropy of an event is given as $f(p) = -p\log_2 p - (1-p)\log_2(1-p)$, where p is the probability of the event. Using the inverse function of $f(p)$, we can calculate the expected hit ratio of the prediction results.

We claim that Branch History Entropy does not depend on any prediction mechanisms. For that reason, Branch History Entropy should be a criterion of branch prediction performance; furthermore, the entropy should offer a theoretical upper bound of prediction performance under a limited condition in which only branch results are used.

The effective criteria of branch prediction performance remain unknown. For that reason, many studies have performed relative comparisons among branch predictors. Discussions were made on certain prediction mechanisms related to the limit of branch prediction performance during the Championship Branch Predictor competitions. Branch History Entropy offers a strong criterion that is independent of prediction mechanisms.

4. ENTROPIES IN BRANCH PREDICTORS

4.1 Branch Predictor Organization

Branch History Entropy quantitatively represents essential information in program behavior, i.e., a string of branch results of taken and not-taken. The entropy can be an important index for representing expected prediction performance independent of branch predictor mechanisms. However, as described in Section 2, the characteristic behaviors of programs appear in imbalanced features in branch predictor mechanisms. This fact demands discussion of essential information involved in predictor organization.

Today's major branch predictors have a common structure, similar to that shown in **Fig. 5**. A predictor inputs some necessary information from the execution part of the processor, such as the program counter (PC) value and branch history. The input information is used to select the proper entry in a table. The table can be considered as a collection of independent prediction functions, and the selected entry does perform an appropriate prediction. After the targeted branch instruction is executed, the predictor is notified whether the prediction was successful or not. The prediction result is reflected into the entry so that the entry can improve its prediction performance.

For example, a bimodal branch predictor[16] inputs the PC value and it selects a table entry using a hashed value of the PC. Each entry in the table consists of a simple sat-

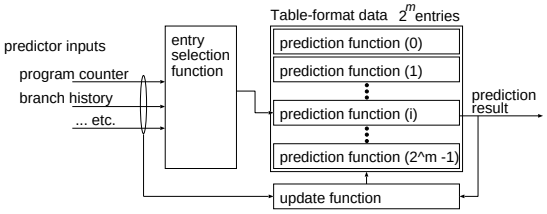


Figure 5: Generalized table-based branch predictor.

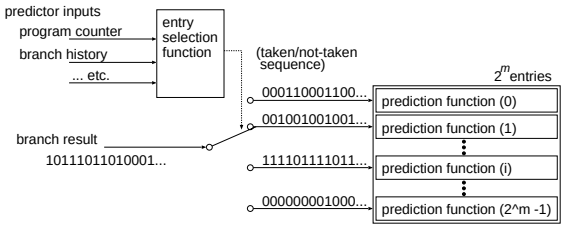


Figure 6: Input sequence of taken and not-taken for each table entry.

uration counter of a few, typically two, bits. At update, if the corresponding branch is taken, the saturation counter is incremented; otherwise, it is decremented.

A two-level branch predictor[22] inputs the latest h bits of branch results, history, and the history points directly to the corresponding entry in the table. Each entry consists of a saturation counter, which is likely to be the bimodal predictor.

The Gshare branch predictor[12] inputs both the PC value and the branch history. It selects a table entry using a hashed value of the result of simple logical operation of the PC and history values, typically, an exclusive-OR operation is used. The method also uses the saturation counter as every table entry.

A perceptron predictor[9, 8] uses a neural network feature of 'perceptrons.' The method inputs the PC value for selecting a table entry. Each table entry comprises a concatenation of *weight* values of the corresponding perceptron. A table entry has h weight values that correspond to h bits of history length and the entry consists of a perceptron neural network that inputs the h-bit branch history. The predictor calculates a weighted sum of each branch history, i.e., $p = \sum_{i=0}^{h-1} w_i \cdot h_i$, where h_i is the i-th history bit and w_i is a corresponding weight value that is a positive or negative numeric. 'Taken' is predicted and otherwise 'not-taken' if the resulting sum becomes positive. The update operation of the predictor is a 'learning' process of the perceptron neural network.

Consequently, today's major branch predictors have roughly two major parts: an entry selection function and a collection of prediction functions. The entry selection function determines which entry is to be used for the subsequent prediction. The selected prediction function performs the actual branch prediction. This interpretation of branch predictor organization suggests two aspects of behaviors: the behavior of the entry selection function and the behavior of each prediction function. We will discuss each of those aspects in the following two subsections.

4.2 Table Reference Entropy

The entry selection function selects an appropriate entry according to its inputs. For that reason, the reference count of each entry is determined using this function. As described in Section 2, reference counts of entries are extremely imbalanced in many programs; the imbalance shows the regularity level of the program.

We introduce the entropy measure also in the imbalanced references of table entries. Assume that a predictor has 2^n entries. Let E_i be i-th entry of the table and r_i be the number of references of E_i. Assuming that the total number of table references is R, and that the occurrence frequency (probability) of the i-th entry E_i is $p(E_i) = r_i/R$, an entropy is defined as

$$H(\overline{R}^n) = -\sum_i p(E_i) \log_2 p(E_i). \quad (4)$$

This quantitative measure represents the imbalance level in table entry references as an entropy. We call the new entropy *Table Reference Entropy* and denote it simply as $H(R)$.

4.3 Table Entry Entropy

According to the logical structure of the generalized table-based branch predictor, only one prediction function is used at a time for prediction of the next branch. Consequently, each prediction function meets a different input sequence of branch results of taken and not-taken. **Fig. 6** shows such a situation.

At each instance of branch prediction, a table entry is selected using the selection function. Simultaneously, a corresponding branch result (taken/not-taken) is served to the selected entry, and the branch result is used to update the entry. For that reason, the served branch results form a sequence *for* individual entry.

The sequence of branch results directly reflects the performance of the corresponding entry, i.e., the prediction hit ratio of the prediction function. For example, consecutive occurrences of the same event, that is to say consecutive taken branches, achieve good performance in a simple saturation counter.

We can also define another entropy measure for representing the regularity in the branch result sequence for each table entry. The entropy is based on Branch History Entropy , which is defined in Section 3, but it is limited for each table entry. We specifically address the i-th entry E_i and present discussion of the limited Branch History Entropy. We can discuss the n-th order augmented adjoint information source $\overline{E_i}^n$ for entry E_i when the string of branch results for the entry is divided into every n consecutive branch results. The n-th order augmented adjoint entropy is given as

$$H(\overline{E_i}^n) = -\sum_k p((E_i^n)_k) \log_2 p((E_i^n)_k), \quad (5)$$

where $(E_i^n)_k$ means the k-th identical symbol of n-consecutive branch results.

The objective entropy of the whole branch predictor is calculated using the weighted sum of the limited Branch History Entropy for all entries. The weight is the occurrence ratio of the entry. For that reason, the resulting entropy

definition is

$$H(\overline{E}^n) = \sum_i p(E_i) H(\overline{E_i}^n), \quad (6)$$

where $p(E_i)$ is the occurrence ratio (probability) of the entry E_i.

Furthermore, we can describe the following relationship similarly to that in Section 3:

$$H^n(E) = H(\overline{E}^{n+1}) - H(\overline{E}^n); \quad (7)$$

we can also reach the true entropy value

$$H(E) = \lim_{n \to \infty} H^n(E). \quad (8)$$

We call the resulting entropy *Table Entry Entropy.*

4.4 Discussion

The term Table Reference Entropy simply represents imbalances in reference counts of table entries. Especially for a two-level branch predictor, Table Reference Entropy approximates Branch History Entropy. For example, a two-level branch predictor uses h bits of branch history and 2^h table entries. The Table Reference Entropy of the predictor represents the imbalance level of h bits of consecutive branch results. The entropy approximately represents the h-th order augmented adjoint source entropy, which is given as Eq. (1). Consequently, only in two-level branch predictor cases, Table Reference Entropy represents the program behavior and branch prediction performance. However, in general, Table Reference Entropy does not represent prediction performance.

In fact, Table Reference Entropy is given as Eq. (4). Its value depends on the number of table entries.

The average information of a string of branch results fed to each table entry is represented by Table Entry Entropy. Because each table entry consists of some prediction function, Table Entry Entropy shows the regularity level of branch results given into each predictor, i.e., the table entry. The value represents the expected prediction performance for the given predictor organization.

Next, we present discussion of the relationship of the proposed three entropies. The necessary information of the forthcoming branch is shown by Branch History Entropy. Suppose that a certain entry E_i is selected. The prediction hit ratio under this certain situation can be considered as a conditional probability of entry E_i. The prediction hit ratio is a simple conditional probability when entry selection and branch execution are mutually independent. They are not always independent in actuality. However, the following relationship is given:

$$H(\overline{B}^n) \leq H(\overline{R}^n) + H(\overline{E}^n). \quad (9)$$

5. EVALUATION

5.1 Preliminary Experiments

As a preliminary evaluation, we measured Branch History Entropy and branch prediction performance using artificially generated branch patterns. For this evaluation, we assume the following. Dominantly executed segments of a program are loops (hot-loops). A program has several hot-loops. Some few paths are executed in loop iterations. For simplification of this discussion, we assume a simple loop structure, which is illustrated in **Fig. 7**. We used the following parameters: the average probability of the taken branch is p_p; the average length of paths in loop iteration is l_p; the number of paths is n_p; and the number of branches between consecutive loop iterations is n_g.

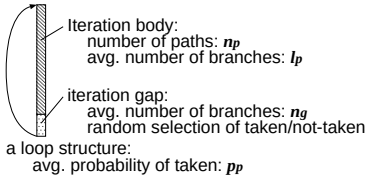


Figure 7: Assumed hot-loop structure and parameters.

Table 4: Parameters for artificial branch pattern generation.

parameter	values
probability of taken branches (p_p):	0.5, 0.6, 0.7, 0.8, 0.9, 0.95
length of paths (l_p):	5, 10, 15, 20, 25, 30
number of paths (n_p):	1, 2, 3, 4, 5, 6, 7, 8
number of gap branches (n_g):	1, 3, 5, 7

Branch result patterns are generated by any possible combination of parameters given in **Table 4**, and Branch History Entropy; the branch prediction performance of the two-level predictor is measured. The length of the generated branch patterns is one million: at least 10 patterns are generated for every combination of the parameter values. The value of Branch History Entropy is assessed using the least-squares method of $H(\overline{B}^{14})$, $H(\overline{B}^{15})$, $H(\overline{B}^{16})$, $H(\overline{B}^{17})$, and $H(\overline{B}^{18})$ values, which is possible because anything but a small variance decreases the accuracy of the measured value. We denote $H_n(B)$ to specify the center value n of the least-squares method explicitly. In this paper, we use $H_{16}(B)$.

Figure 8 shows the Branch History Entropy $H_{16}(B)$ and the prediction hit ratio combination for each generated branch pattern.

Branch History Entropy represents an entropy metric of the next branch. As described in Section 3.3, we can calculate the expected hit ratio of prediction. Information entropy is given as $f(p) = -p \log_2 p - (1-p) \log_2 (1-p)$ when a system has binary events (0 or 1) and one event occurs at a given probability p. By calculating the inverse operation of $f(p)$, $f^{-1}(p)$, we can obtain the probability p from a measured entropy value. The probability represents the expected prediction hit ratio.

Figure 9 shows the expected hit ratio, as calculated from $H_{16}(B)$ and the actual prediction hit ratio. The solid line shows $y = x$.

At some points, the prediction hit ratio is greater than the expected hit ratio in Fig. 9. The actual n-th order Branch History Entropy $H(\overline{B}^n)$ is not a monotonic function for n. Sometimes this causes errors in the calculated $H_n(B)$ value; and the error underestimates the hit ratio.

The resultant prediction hit ratio is widely distributed when the expected hit ratio is low. Actually, Branch History Entropy is independent of the organization and mech-

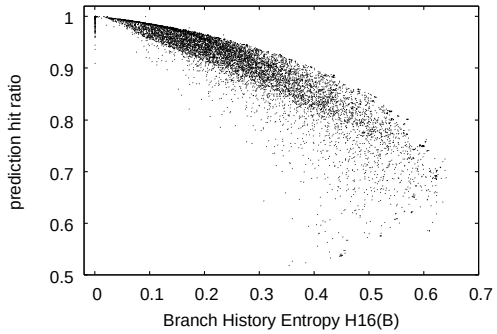


Figure 8: Branch History Entropy ($H_{16}(B)$) and the prediction hit ratio under artificially generated branch patterns.

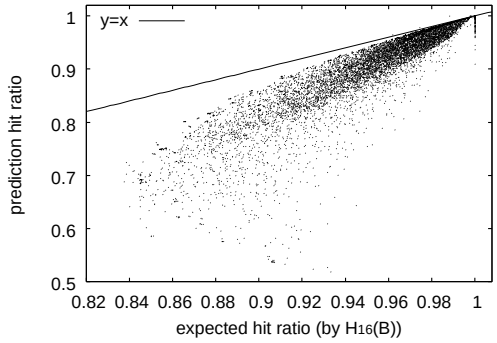


Figure 9: Expected hit ratio and the prediction hit ratio under artificially generated branch patterns.

anism of the predictor. In actuality, the two-level predictor uses two-bit saturation counters. The prediction mechanisms cannot keep track of the branch behavior, even when the behavior is perfectly predictable. For example, a certain branch pattern, say `0101...`, is completely predictable; however, the predictor with two-bit saturation counter yields only 50% of hits.

5.2 CBP2 Results

We next performed more practical evaluations using the framework prepared for the second Championship Branch Predictor (CBP2 [20]). We implemented a measurement function for Branch History Entropy in the CBP2 framework. Branch History Entropy was measured in every one million branches.

Actually, **Table 5** shows the expected prediction hit ratio given as $H_{16}(B)$, and *idealistic prediction hit ratio* of GTL and PMPM from CBP2 results[20]. The idealistic prediction hit ratio is not *ideal* one. It shows the maximum expectation when no hardware limitation is on the predictor. **Figure 10** depicts every combination of expected and idealistic hit ratios given in Table 5. Dotted lines in the figure show linear expressions of fitting results using least-squares method.

5.3 SPEC CPU2000 Results

The CBP2 framework offers trace results for discussing branch prediction techniques. Therefore, the amounts of traces are insufficient for statistical discussion. For that reason, we measured our defined entropies and predictor performance metrics on SPEC CPU2000[17] benchmark programs using SimpleScalar tool-set[15].

Table 5: Comparison of the expected prediction hit ratio based on $H_{16}(B)$ and idealistic hit ratio of CBP2 results.

benchmark	num. of pred.	expected hit ratio	GTL	PMPM
164.gzip	15,114,618	0.9659	0.9379	0.9373
175.vpr	12,827,240	0.9503	0.9393	0.9320
176.gcc	17,083,265	0.9702	0.9858	0.9848
181.mcf	21,700,442	0.9723	0.9663	0.9646
186.crafty	8,927,054	0.9721	0.9822	0.9799
197.parser	14,190,768	0.9647	0.9726	0.9723
201.compress	11,764,907	0.9677	0.9553	0.9561
202.jess	14,154,077	0.9932	0.9979	0.9980
205.raytrace	12,201,887	0.9717	0.9981	0.9978
209.db	13,191,792	0.9827	0.9836	0.9833
213.javac	12,986,612	0.9845	0.9927	0.9928
222.mpegaudio	13,021,228	0.9844	0.9928	0.9929
227.mtrt	12,512,710	0.9721	0.9978	0.9975
228.jack	11,926,294	0.9826	0.9964	0.9960
252.eon	7,724,982	0.9800	0.9977	0.9970
253.perlbmk	13,537,140	0.9912	0.9991	0.9986
254.gap	13,838,063	0.9824	0.9918	0.9922
255.vortex	11,073,783	0.9811	0.9992	0.9992
256.bzip2	24,586,298	0.9996	0.9999	0.9999
300.twolf	13,098,915	0.9348	0.9247	0.9196

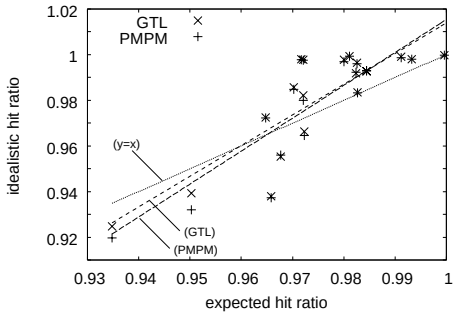


Figure 10: Comparison of the expected hit ratio (by $H_{16}(B)$) with the CBP idealistic hit ratio.

Our proposed three entropies, Branch History Entropy, Table Reference Entropy and Table Entry Entropy, and prediction performance (hit ratio) are measured in every time window. The length of the time window is one million conditional branches. All of our evaluation metrics are based on the number of conditional branches. For that reason, other time window measures, such as the clock cycles and the number of instructions executed, are unsuitable for our evaluation purposes. For example, different program executions have different numbers of conditional branches in a certain period in clock cycles.

We should consider so-called 'phased behavior' for a proper time window. A major concern in this paper is program behavior. Consequently, we should not flatten temporal and characteristic features of program behavior. The time window offers granularity in representing the behavior.

Another important item in discussing the time window is accuracy. The narrow time window offers a small number of samples. Entropies are measured statistically. They fundamentally require a sufficient number of samples. The use of fewer samples engenders less confidence in measured entropies.

In this paper, we used one million conditional branches for the evaluation time window in reference to other research results in program behavior[9, 8].

5.3.1 Time sequence plots of measured parameters

Most programs change their execution 'phases' according to their execution situation. **Figure 11** presents time sequence plots of the prediction hit ratio, Branch History Entropy $H_{16}(B)$, Table Reference Entropy $H(R)$, and Table Entry Entropy $H_{16}(E)$. In Fig. 11, Figs. 11(a)– 11(c) graphs are of integer applications; and Figs. 11(d)– 11(f) are floating-point applications. Figures 11(a) and 11(d) show some rhythms, Figs. 11(b) and 11(e) show irregular behavior, and Figs. 11(c) and 11(f) are stable.

For this evaluation, the two-level branch predictor with 16-bit history length is used. Every graph in Fig. 11 has the same y-axis scale, and scaling of the x-axis is different because of the different execution time length.

Each graph gives a perspective view of the corresponding program behavior and predictor performance. Roughly speaking, prediction performance curves are symmetric to those of $H_{16}(B)$ and $H(R)$ entropies. Actually, Table Entry Entropy $H_{16}(E)$ does not show a strong correlation with other metrics, the hit ratio, and Branch History and Table Reference Entropies, in many programs.

We further discuss individual entropy in the following subsections.

5.3.2 Branch History Entropy

Figure 12 presents the expected prediction hit ratio and the actual prediction hit ratio. Each dot in the graph shows a measured combination of the expected prediction hit ratio and the actual prediction hit ratio in a certain time window. The solid line shows $y = x$.

We can find that prediction performance is strongly correlated using Branch History Entropy. In fact, Branch History Entropy represents the regularity level of program behavior that is independent of the prediction method. Simply speaking, a highly regular program produces a regular sequence of branch results. Therefore, predictors can produce accurate predictions.

Both Gshare and perceptron predictors sometimes achieve better performance than the expected hit ratio, as shown in Figs. 12(c) and 12(d). These points explicitly represent the merits of a prediction mechanism.

5.3.3 Table Reference Entropy

Figure 13 shows Table Reference Entropy and prediction hit ratios, similarly to the Branch History Entropy and prediction hit ratio plots in Fig. 12. The maximum possible value of $H(R)$ depends on the number of table entries in the predictor. The number is determined using the same hardware-budget rule. For that reason, each predictor has a different number of entries.

In fact, Table Reference Entropy does not show prediction performance itself, but it shows the number of entries that are accessed. Therefore, the correlation between Table Reference Entropy and the prediction hit ratio is weak, as shown in the figure. As discussed in Section 4.4, Table Reference Entropy of two-level branch predictor approximates Branch History Entropy. Consequently, the two-level predictor results in Fig. 13(b) show correlation to some degree.

5.3.4 Table Entry Entropy

Figure 14 shows the expected prediction performance derived using Table Entry Entropy $H_{16}(E)$ and the actual prediction hit ratio, similarly to the depiction in Fig. 12.

Each dot represents a measured combination of expected and actual prediction hit ratios. From Eq. (9), because Table Reference Entropy $H(R)$ has a positive value in general, Table Entry Entropy is lower than Branch History Entropy. For that reason, the expected prediction hit ratio based on Table Entry Entropy is larger than that based on Branch History Entropy. The steep fitting lines indicate a loss of information which might be used for accurate prediction.

5.4 Discussion

Figure 12(b), which illustrates the correlation between the expected prediction hit ratio from Branch History Entropy and the actual hit ratio, roughly matches with about one quarter in the upper left part of Fig. 9. Figure 9 shows the correlation in artificially generated patterns, whereas Fig. 12(b) shows practical ones. This result supports our evaluation results.

Distribution patterns in Figs. 12(b)– 12(d) are very similar, but their prediction methods differ greatly, i.e., two-level, Gshare, and perceptron. This shows the appropriateness of our claim that Branch History Entropy is a suitable criterion for representing the predictor performance.

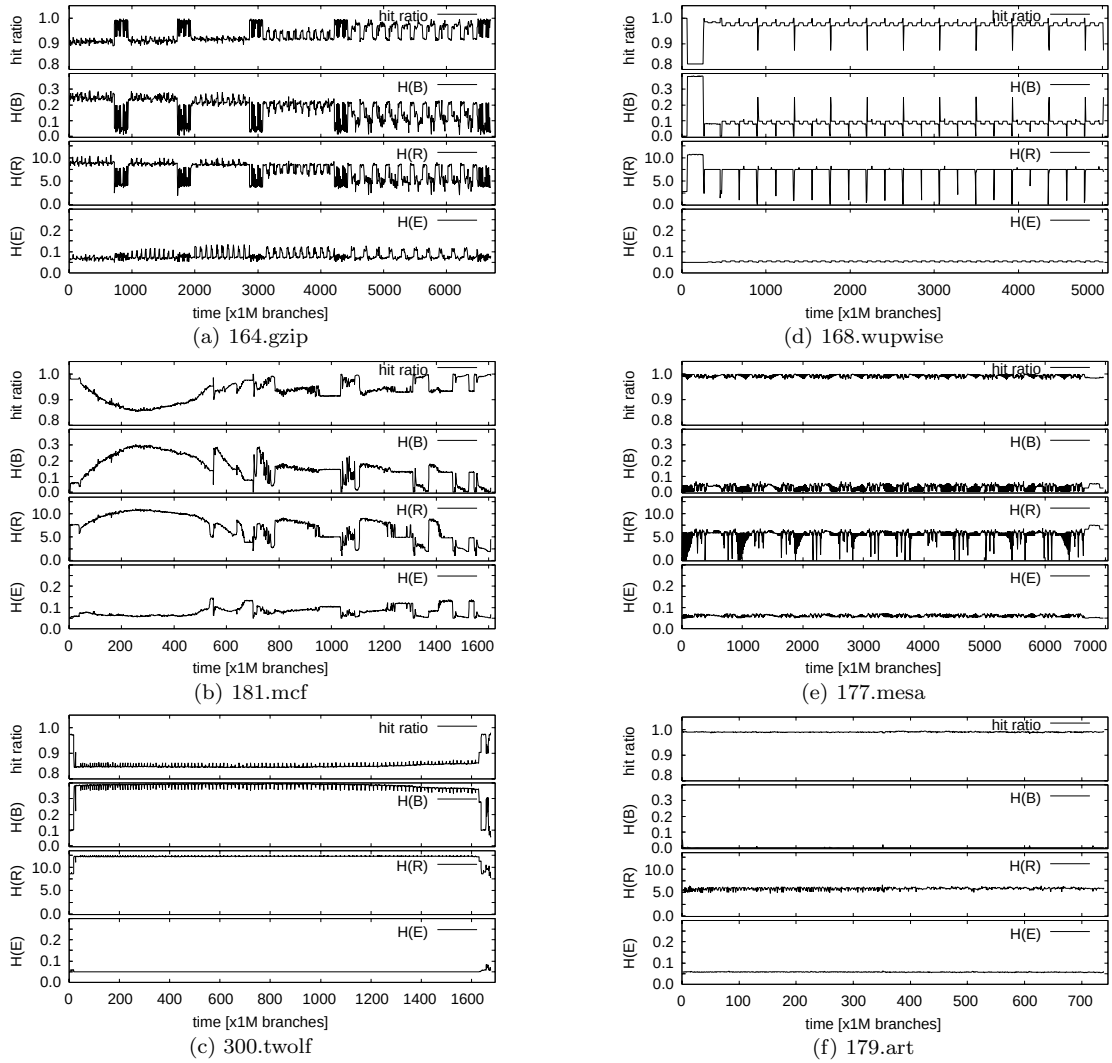
Table Entry Entropy shows the maximum expectation in prediction performance under the table-based organization. As shown in Fig. 14, most actual prediction hit ratios are much lower than the expected hit ratios because no table entry fully utilizes the information involved in the incoming branch results. For example, bimodal, two-level, and Gshare predictors use a small saturation counter at each table entry. Such counters do not perform full use of information.

As another example, the perceptron predictor achieves good performance (Fig. 14(d)), which implies the effect of 'learning,' accumulated in weight values stored in each table entry.

All the predictors shown in Fig. 14 have steep fitting lines, meaning that the branch prediction performance can be improved greatly if no hardware limitation exists.

6. RELATED WORK

In general, predictors (sometimes implicitly) subsume imbalanced features in executing a program. For example, when a predictor has some possible events, it selects the most frequently occurred event among the possible ones. It might predict the same event as the most recent one. Prediction performance of such predictors is clearly dependent on the imbalance (or regularity) level of event generation. Consequently, many studies specifically address the efficient extraction of imbalances.



(a) 164.gzip (d) 168.wupwise

(b) 181.mcf (e) 177.mesa

(c) 300.twolf (f) 179.art

Figure 11: Time sequence plots of the prediction hit ratio, $H_{16}(B)$, $H(R)$, and $H_{16}(E)$ in SPEC CPU2000 benchmarks.

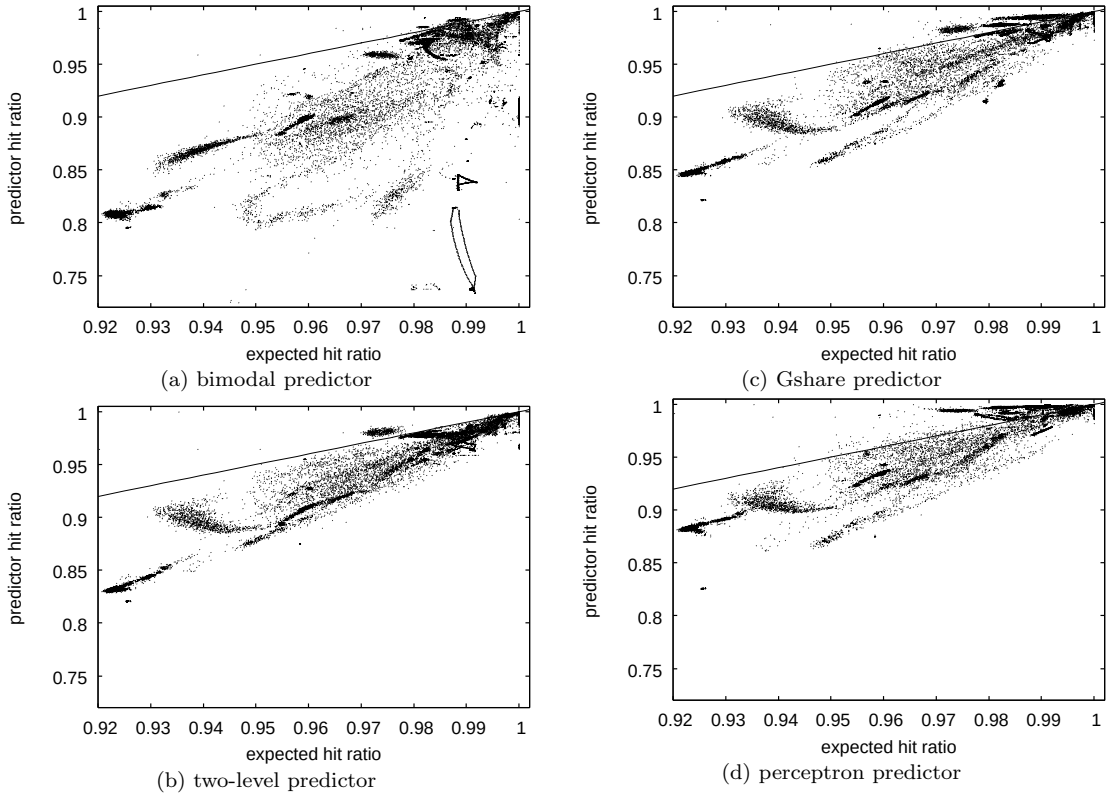
Kise et al.[11] address extremely biased branches and discuss a new predictor. Tyson et al.[21] investigated the sequence of conditional branch results, i.e. taken and not-taken, and some imbalanced features in the sequences of taken and not-taken results. They show four typical cases of the taken and not-taken sequences; long consecutive takens, long consecutive not-takens, a small number of not-takens in long consecutive takens, and other patterns. Such classification increases the prediction performance in their methods. Their results are evaluated as prediction hit ratios by simulation. However, they do not describe the degrees of imbalance (or regularity) and their influence on prediction performance.

Jacobsen et al.[6] propose confidence metrics for branch prediction results. Another study proposes a confidence estimation mechanism using a neural network (perceptron)[1]. The concept of 'confidence' might appear to be similar to our entropies, but it is definitely different. Confidence gives the degree of likelihood of the prediction result. Branch History Entropy quantitatively represents regularity in program execution. Consequently, the entropy can also represent the theoretical predictor performance, independent of prediction methods and mechanisms.

Another aspect of view of regularity is 'periodicity.' Dynamic Periodicity Detector (DPD), which was proposed by Freitag et al.[5], specifically examines sequences of data values that appear in program execution. As another example of periodicity, the Fourier Analysis Branch (FAB) predictor, proposed by Kampe et al.[10], uses the string of taken/not-taken results of branch instructions and extracts periodicities through Fourier transformation. By extrapolation of the curve of taken/not-taken results, using the resulting Fourier coefficients, the predictor predicts the next event.

Fourier analysis results show some regularity levels of program behavior. However, they solely emphasize periodicity. Furthermore, our proposed entropies quantitatively represent regularity levels in one value, whereas Fourier analysis requires a group of values to represent the regularity.

Mudge and Chen et al.[2, 13] present limits in the (upper bound of) prediction performance based on prediction using



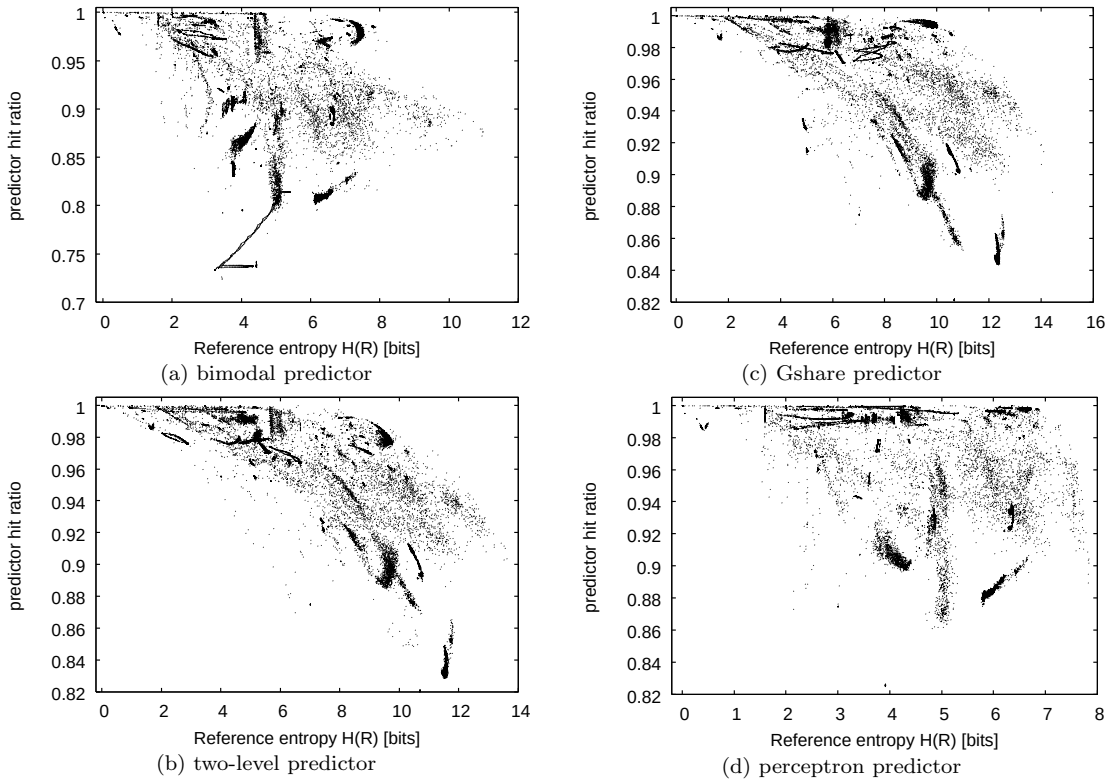
(a) bimodal predictor

(c) Gshare predictor

(b) two-level predictor

(d) perceptron predictor

Figure 12: Expected prediction hit ratio from $H_{16}(B)$ and the actual hit ratio.



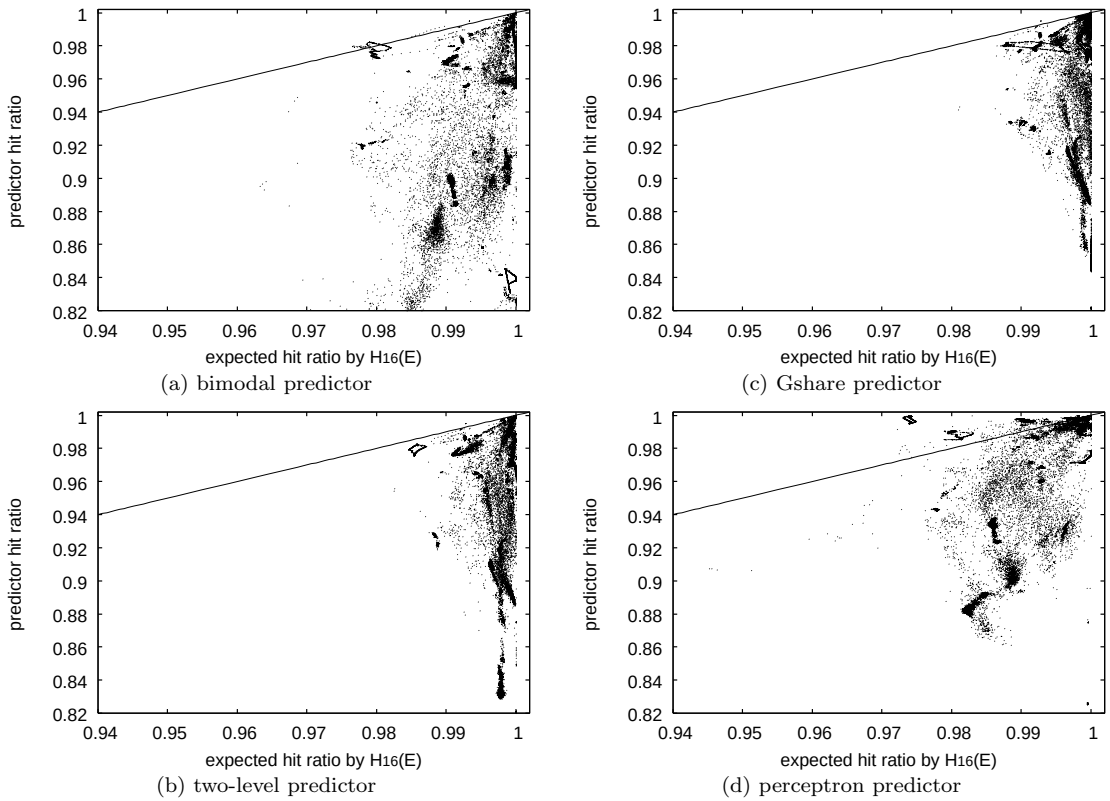
(a) bimodal predictor

(c) Gshare predictor

(b) two-level predictor

(d) perceptron predictor

Figure 13: Table Reference Entropy $H(R)$ and prediction hit ratio.



(a) bimodal predictor

(c) Gshare predictor

(b) two-level predictor

(d) perceptron predictor

Figure 14: Expected prediction hit ratio from $H_{16}(E)$ and the actual hit ratio.

the Partial Matching (PPM) method. Actually a data compression algorithm, PPM was originally developed for image processing, and it is proved that the method is theoretically optimal. They use m-th order Markov predictor based on an m-bit length of branch history. The performance of the m-th order Markov predictor corresponds to our proposed m-th order augmented adjoint Branch History Entropy, as described in Eq. (1). Their discussion emphasizes prediction performance at a theoretically ideal situation. However, our proposed method offers quantitative representation. Driesen et al.[3, 4] discuss limits of indirect branch prediction from a different point of view from that used in this paper.

Jha et al.[7] also use a Markovian model to represent an optimal prediction mechanism for a given history length. Their major subject is to obtain a practically possible prediction function using a Moore automaton that represents state transitions in a predictor. However, their Markovian predictor does not address imbalanced (or regular) features in program execution, as our defined entropies do. Furthermore, their main objective is not the quantitative representation of program behaviors and predictor performances.

Our approach is unique in its theoretical and quantitative approach based on information entropy.

7. CONCLUSIONS

Predictors are inherent in high-performance microprocessors. Numerous studies have reported *new* methods for them. Among the many types of application programs, none have a satisfactory predictor. For that reason, even when a predictor A achieves better prediction performance than that of a predictor B in certain programs, B might perform better than A in another kind of application programs. Actually, a perceptron predictor performs well in most programs, but some programs show that the Gshare predictor performs better.

We know empirically that regular behavior induces good prediction performance. On the other hand, irregular (or randomized) behavior loses predictability. Some degree of such 'regularity', if we can define it, should be a useful criterion for prediction performance. This basic idea motivated us to examine program behavior and prediction performance.

Our approach is based on information entropy. Information entropy quantitatively represents the regularity and randomness level. We first described Shannon's discussion on information entropy; we then applied the same idea to branch predictors. That is, the microprocessor executing a program can be considered as an information source that generates a sequence of taken/not-taken results of branch instructions. Through its application, we reached a new entropy, Branch History Entropy, which quantitatively represents program behavior independent of predictor mechanisms. The entropy also shows the expected prediction performance (hit ratio).

We further pursued the entropy idea into predictor organization. We specifically described a table-based branch predictor, and showed a stereoscopic view with two aspects of entropies, Table Reference Entropy and Table Entry Entropy. The former shows an imbalanced number of refer-

ences on table entries. The latter shows net regularity in each table entry. It is important for discussing the theoretical upper bound of prediction performance.

We evaluated our proposed entropies and prediction performance in three aspects: artificially generated branch patterns, Championship Branch Predictor (CBP2) competition materials, and SPEC CPU2000 benchmark programs. Evaluation results show the important perspectives of the proposed entropies. Branch History Entropy is an appropriate criterion for comparing predictor methods. Table Reference Entropy depends strongly on prediction methods and it does not always represent prediction performance, but it shows resource efficiencies. The actual prediction performance is far worse than the expected hit ratio induced from Table Entry Entropy. Consequently, Table Entry Entropy offers theoretical upper bound performance of the table-based predictor, and its high potential remains for improving the prediction performance.

The major contribution in this paper is provision of theoretical and quantitative indices of program behavior and prediction performance. Our evaluation results reveal the proposed entropies' high potential. They should be important criteria for discussing the predictor architecture.

8. ACKNOWLEDGMENTS

The authors thank anonymous reviewers for their valuable comments and suggestions. This research was supported in part by Grant-in-Aid for Scientific Research ((B) 18300014, (C)16500023, 19500037) and Young Scientists ((B)17700047) of Japan Society for the Promotion of Science (JSPS), and by Eminent Research Selected at Utsunomiya University.

9. REFERENCES

[1] H. Akkay, S. T. Srinivasan, R. Koltur, Y. Patil, and W. Refaai. Perceptron-based branch confidence estimation. In *Proc. 10th Int'l Symp. High-Performance Computer Architecture*, pages 265–274, 2004.

[2] I.-C. K. Chen, J. T. Coffey, and T. N. Mudge. Analysis of branch prediction via data compression. In *Proc. 7th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pages 128–137, Oct. 1996.

[3] K. Driesen and U. Hölzle. Limits of indirect branch prediction. Technical Report TRCS97–10, Computer Science Department, University of California, Santa Barbara,, June 1997.

[4] K. Driesen and U. Hölzle. Multi-stage cascaded prediction. Technical Report TRCS99–05, Computer Science Department, University of California, Santa Barbara,, Feb. 1999.

[5] F. Freitag, J. Corbalan, and J. Labarta. A dynamic periodicity detector: Application to speedup computation. In *Proc. 15th Int'l Parallel and Distributed Processing Symp.*, Apr. 2001.

[6] E. Jacobsen, E. Rotenberg, and J. E. Smith. Assigning confidence to conditional branch predictions. In *Proc. 29th Annual IEEE/ACM Int'l Symp. Microarchitecture*, pages 142–152, 1996.

[7] S. Jha, Y. Lu, and E. Clarke. Formal analysis of branch prediction algorithm. Technical report, Computer Science, Carnegie Mellon University, 1998.

[8] D. A. Jiménez. Piecewise linear branch prediction. In *Proc. 32nd Annual Int'l Symp. Computer Architecture*, pages 382–393, 2005.

[9] D. A. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *Proc. 7th Int'l Symp. High-Performance Computer Architecture*, pages 197–206, Jan. 2001.

[10] M. Kampe, P. Stenstrom, and M. Dubois. The fab predictor: Using fourier analysis to predict the outcome of conditional branches. In *Proc. Eighth Int'l Symp. High-Performance Computer Architecture*, pages 223–232, Feb. 2002.

[11] K. Kise, T. Katagiri, H. Honda, and T. Yuba. The bimode++ branch predictor using the feature of extremely biased branches. *IPSJ SIG Tech. Report*, 2005(7):57–62, Jan. 2005.

[12] S. McFarling. Combining branch predictors. Technical Report TN–36, Digital Equipment Corp., Western Research Laboratory, June 1993.

[13] T. Mudge, I.-C. Chen, and J. Coffey. Limits to branch prediction. Technical Report CSE–TR–282–96, University of Michigan,, Feb. 1996.

[14] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, July and October 1948.

[15] SimpleScalar LLC. http://www.simplescalar.com/.

[16] J. E. Smith. A study of branch prediction strategies. In *Proc. 8th Int'l Symp. Computer Architecture*, pages 135–148, May 1981.

[17] Standard Performance Evaluation Corporation. SPEC CPU2000 V1.3. http://www.spec.org/cpu2000/.

[18] Standard Performance Evaluation Corporation. SPEC CPU95 Benchmarks. http://www.spec.org/cpu95/.

[19] The 1st JILP Championship Branch Prediction Competition. http://www.jilp.org/cbp/, 2004.

[20] The 2nd JILP Championship Branch Prediction Competition. http://camino.rutgers.edu/cbp2/, 2006.

[21] G. Tyson, K. Lick, and M. Farrens. Limited dual path execution. Technical Report CSE–TR–346–97, University of Michigan, 1997.

[22] T.-Y. Yeh and Y. N. Patt. Two-level adaptive branch prediction. In *Proc. 24th ACM/IEEE Int'l Symp. Microarchitecture*, pages 51–61, Nov. 1991.

[23] T. Yokota, K. Ootsu, F. Furukawa, and T. Baba. An entropy measure for branch predictor performance. *IPSJ SIG Tech. Report*, 2005(120):81–86, Dec. 2005.

[24] T. Yokota, K. Ootsu, F. Furukawa, and T. Baba. Entropy properties in program behaviors and branch predictors. In *Proc. 18th IASTED Int'l Conf. Parallel and Distributed Computing and Systems*, pages 448–453, Nov. 2006.

[25] T. Yokota, M. Saito, F. Furukawa, K. Ootsu, and T. Baba. Two-path limited speculation method for static/dynamic optimization in multithreaded systems. In *Proc. 6th Int'l Conf. Parallel and Distributed Computing, Applications and Technologies*, pages 46–50, Dec. 2005.

[26] T. Yokota, M. Saito, K. Ootsu, F. Furukawa, and T. Baba. Two-path limited speculation method. *IPSJ Trans. Advanced Computing Systems*, 46(SIG 16 (ACS–12)):1–13, Dec. 2005.