Journal of Instruction-Level Parallelism 5(2003) 1-32 Submitted 2/02; published 6/03

Exploiting Bias in the Hysteresis Bit of 2-bit Saturating Counters in Branch Predictors

Gabriel H. Loh[†] LOH@CS.YALE.EDU
Dana S. Henry[‡†] DANA.HENRY@YALE.EDU
Arvind Krishnamurthy[†] ARVIND.KRISHNAMURTHY@YALE.EDU
[†]*Department of Computer Science, Yale University*
[‡]*Department of Electrical Engineering, Yale University*
New Haven, CT, 06520-8285

Abstract

The states of the 2-bit counters used in many branch prediction schemes can be divided into "strong" and "weak" states. Instead of the typical saturating counter encoding of the states, the 2-bit counter can be encoded such that the least significant bit directly represents whether the current state is strong or weak. This extra bit provides hysteresis to prevent the counters from switching directions too quickly. Past studies have exploited the strong bias of the direction bit to construct better branch predictors. We show that counters exhibit a strong bias in the hysteresis bit as well, suggesting that an entire bit dedicated to hysteresis is overkill. Using data-compression techniques, we empirically demonstrate that the information theoretic entropy of the hysteresis bit conveys less than 0.18 bits per prediction of information for a gshare *branch predictor. We explain how to construct fractional-bit* shared split counters *(SSC) by sharing a single hysteresis bit between multiple counters. We show that predictors implemented with shared split counters perform nearly as well as the corresponding two-bit counter versions, while providing area reductions of 25-37.5%.*

1. Introduction

Ever since the saturating 2-bit counter was introduced for dynamic branch prediction, it has been the default finite state machine used in most branch predictor designs. Smith observed that using two bits per counter yields better predictor performance than using a single bit per counter, and using more than two bits per counter does not improve performance any further [1]. The question this study addresses is somewhat odd: does a two-bit counter perform much better than a k-bit counter, for $1 < k < 2$? If not, the size of the branch predictor can be reduced to $\frac{k}{2}$ of its original size. This naturally leads to asking if, for example, a 1.4-bit counter even makes any sense. We do not actually design any 1.4-bit counters, but instead we propose counters that have fractional costs by sharing some state between multiple counters.

Each bit of the two-bit counter plays a different role. The most significant bit, which we refer to as the *direction bit*, tracks the direction of branches. The least significant bit provides hysteresis which prevents the direction bit from immediately changing when a misprediction occurs. The Merriam-Webster dictionary's definition of hysteresis is "a retardation of an effect when the forces acting upon a body are changed," which is a very accurate description of the effects of the second bit of the saturating two-bit counter. We refer to the least significant bit of the counter as the *hysteresis bit* throughout the rest of this paper.

Although the hysteresis bit of the saturating two-bit counter prevents the branch predictor from switching predicted directions too quickly, if most of the counters stay in the strongly taken or strongly not-taken states most of the time, then perhaps this information can be shared between more than one branch without too much interference. In this study, we examine how strong the biases of the hysteresis bits of the branch prediction counters are, and then use this information to motivate the design of more compact branch predictors.

Although the trend in branch predictor design appears to be toward larger predictors for higher accuracy, the size of the structures can not be ignored. The gains from higher branch prediction accuracy can be negated if the clock speed is compromised [2]. Our shared split counters may enable the reduction of the area requirements of branch predictors, which leads to shorter wire lengths and decreased capacitative loading, which in turn may result in faster access times. Compact branch prediction structures may also be valuable in the space of embedded processors where smaller branch prediction structures use up less chip area and require less power.

The rest of this paper is organized as follows. Section 2 provides a brief overview of basic information theory and discusses some related research in branch prediction. Section 3 presents an analysis of the bias of the hysteresis bit in branch prediction counters. Section 4 describes the hardware organization for our shared split counters as well the performance results. Section 5 classifies the types of mispredictions that arise due to shared hysteresis bits and explains why the sharing does not induce very many additional mispredictions. Finally, Section 6 draws some final conclusions.

2. Background

In this paper, we use an information theoretic approach to analyze the patterns and behavior of the hysteresis bit in branch predictors. The first part of this section provides a brief review of some basics in information theory. The remainder of the section provides additional context for this paper by discussing related branch prediction research.

2.1 A Brief Information Theory Primer

Information theory was first developed by Claude Shannon to address the issues of encoding signals for transmission over noisy channels. The theory addresses the information of the signal source, the information capacity of the channel, and how to encode the message to match the capacity of the channel. The term *information* is used in its technical sense, which we will define below. In the context of this paper, and in information theory in general, the term information should not be confused with difficult to define terms like “knowledge”. We will first present an informal, qualitative discussion on information, and then proceed to the technical definitions.

We will use a few simple sentences to provide an example of different *messages* that convey different amounts of information. Consider the following sentences:

1. I woke up today.
2. I went to work today.
3. I won the lottery today.

If someone were to state the first sentence, it would not be very surprising to hear, and in some sense conveys very little information because waking up is an action that almost every living person

performs everyday. The second sentence may provide some additional information, because if the person went to work, then it is likely that today is not Saturday or Sunday. The last sentence conveys a large amount of information because winning the lottery is an event that is not likely to happen to someone on a regular basis. From these examples, we can see that our intuitive notion of information is correlated with the uncertainty or probability of the occurrence of an event. In terms of information theory, we are more interested in *messages* than events, but the same idea applies: messages with low probabilities convey more information.

The formal definition of a message's information depends on the probability of that message. Assume a message x_i has a probability of $P(x_i) = P_i$ of occurring. Then, I_i is the information or *self-information* of the message:

$$I_i = -\log_b P_i = \log_b \frac{1}{P_i}$$

where b is the logarithmic base. The units of information depends on the choice of b, although typically $b = 2$ which gives the information in units of bits. Shannon chose the logarithmic function because it is the only function that satisfies the following properties:

1. The information is never negative ($I_i \geq 0$ for $P_i \in [0, 1]$).
2. The information tends to zero as the probability of the message becomes a certainty ($\lim_{P_i \to 1} I_i = 0$).
3. The less likely that a message occurs, the more information the message will convey ($I_i > I_j$ for $P_i < P_j$).
4. The information conveyed by two independent messages is equal to the sum of the information of the individual messages ($I_{ij} = \log_b \frac{1}{P_i P_j} = \log_b \frac{1}{P_i} + \log_b \frac{1}{P_j} = I_i + I_j$ when $P(x_i x_j) = P(x_i)P(x_j)$).

In computer engineering, the "messages" that are usually used are the two symbols "0" and "1". Note that if each of these two occur with a equal probability of $P_0 = P_1 = \frac{1}{2}$, then the information of each symbol is $I_0 = I_1 = \log_2 \frac{1}{1/2} = \log_2 2 = 1$ bit. A binary digit (bit) can represent any set of two messages, but this should not be confused with a bit of *information* since a particular zero or one may represent more or less than a single bit of information when the probabilities are not uniform.

A source X that emits a sequence of messages or symbols has a corresponding *entropy* or information per symbol. Assuming the source generates successive symbols with independent probabilities, the total amount of information for a sequence of k symbols $x_1, x_2, ..., x_k$ is the sum of the individual information $\sum_{i=1}^{k} x_i$. As the number of symbols increases, $k \gg 1$, the expected information per symbol is equal to

$$H(X) = \sum_{i=1}^{k} P_i I_i = \sum_{i=1}^{k} P_i \log_2 \frac{1}{P_i}$$

where $H(X)$ is the *entropy* of the source X, and is given in units of bits per symbol. Note that the *information rate* of X is a measure of the bits per second, and so is equal to the entropy times the symbol generation rate.

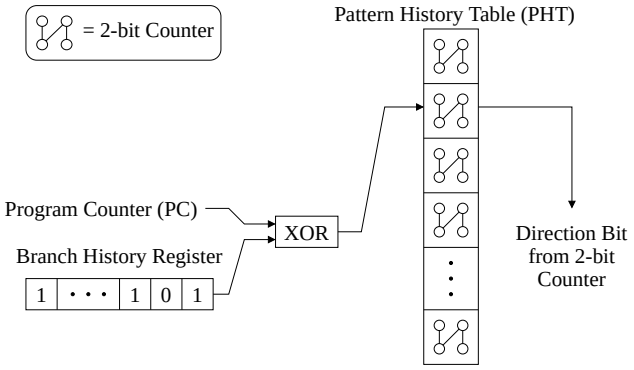

Figure 1: The gshare branch predictor combines both the program counter and global branch history to index into the pattern history table.

In this research, we use the ideas of information theory to measure the entropy of the hysteresis bits in branch predictor counters. The results motivate a predictor structure that uses fewer bits to encode approximately the same information that is represented by the original two-bit saturating counter organization.

2.2 Related Branch Prediction Research

Hysteresis in dynamic branch predictors reduces the number of mispredictions caused by *anomalous decisions*. If a particular branch instruction is predominantly biased in the taken direction for example, a single bit of state recording the branch's most recent direction will make two mispredictions if a single not-taken instance is encountered (for example, at a loop exit). The first misprediction is due to the anomalous decision, and the second misprediction is due to the previous branch throwing off the recorded direction of the branch. The most common mechanism to avoid this extra misprediction is the saturating 2-bit counter introduced by Smith [1].

Smith's branch predictor maintains a table of 2-bit counters indexed by the branch address. Pan et al [3] and Yeh and Patt [4] [5] [6] studied how to combine branch outcome history with the branch address to correlate branch predictions with past events. The index is formed by concatenating n bits of the branch address with the outcomes of the last m branch instructions. This index of length $m + n$ is used to look up a prediction in a 2^{m+n}-entry table of 2-bit counters.

The prediction schemes presented by Yeh and Patt use a table of 2^{m+n} two-bit counters. To prevent the table from becoming unreasonably large, the sum $m + n$ must be constrained. This forces the designer to make a tradeoff between the number of branch address bits used, which differentiate static branches, and the number of branch history bits used, which improve prediction performance by correlating on past branch patterns. McFarling proposed the *gshare* scheme that makes better use of the branch address and branch history information [7]. Figure 1 illustrates how the global branch history is xor-ed with the program counter to form an index into the pattern history table (PHT). The most significant bit of the 2-bit counter in the indexed PHT entry, the direction bit, is used for the branch prediction. With this approach, many more unique $\{$branch address, branch history$\}$ pairs may be distinguished, but the opportunities for unrelated branches to map to the same two-bit counter also increase. Much research effort has addressed the interference in gshare styled

predictors [8] [9] [10] [11] [12] [13]. Many of these predictors exploit the fact that the direction of the counters in the table are strongly biased in one direction or the other.

One such approach is the *agree* predictor [13]. Sprangle et al make the observation that the direction of most branches is highly biased. The agree predictor takes advantage of this by storing the predicted direction outside of the counter in a separate *biasing bit*, and then reinterpreting the two-bit counter as an "agreement predictor" (i.e. do the branch outcomes agree with the biasing bit?). The biasing bit is stored in the branch target buffer (BTB), and is initialized to the first outcome of that particular branch. This scheme reduces some of the negative effects of interference by converting branches that conflict in predicted branch direction to branches that agree with their bias bits.

The Bi-Mode algorithm is another predictor that reduces interference in the PHT [10]. A *choice PHT* stores the predominant direction, or *bias* of the branch (the bias in the context of the Bi-Mode predictor is a separate concept from the biasing bit of the agree predictor). The bias is then used to select one of two *direction PHTs*. The idea is that branches with a taken bias will be sorted into one PHT, while branches with a not-taken bias will be sorted into the other PHT. If interference occurs within a direction PHT, the branches are likely to have similar biases, thus converting instances of destructive aliasing into neutral interference (see Section 4.2, Figure 7 for a diagram of the Bi-Mode predictor).

The gskewed predictor takes a voting-based approach to reduce the effects of interference [12]. Three different PHT banks are indexed with three different hashes of the branch address and branch history. A majority vote of the predictions from each of the PHTs determines the overall prediction. With properly chosen hash functions, two $\{$branch address, branch history$\}$ pairs that alias in one PHT bank will not conflict in the other two, thus allowing the majority function to effectively ignore the vote from the bank where the conflict occurred.

The branch predictor designed for the terminated EV8 processor used a gskewed derived hybrid predictor [14]. The first gskewed PHT bank, called *BIM* for bimodal, only makes use of the branch address for indexing where as the other two banks, G0 and G1, use an exclusive-or of the branch address and branch history. An additional bank, *Meta*, decides whether the final prediction should come from the BIM bank, or from the majority vote of all three banks. Of particular interest in relation to this paper, the EV8 hysteresis arrays for the Meta and G1 banks only use half the number of entries than in the direction arrays. In this paper, our predictor uses a slightly different finite state machine, we explore more aggressive reductions in the hysteresis array size, and we provide an analysis for why sharing hysteresis bits between multiple counters does not have a large impact on prediction accuracy.

3. How Many Bits Does it Take?

3.1 Branch Prediction Counters

An important result of the agree predictor study [13] is that the branch direction and other dynamic predictor state can be separated. One of the reasons that the agree predictor is able to reduce the effects of destructive interference is that although there are aliasing conflicts for some of the predictor's state (namely the agree counters), this conflict may not impact the prediction of a particular branch if there is no aliasing of the recorded branch directions.

The two-bit counter is simply a finite state machine with state transitions and encodings that correspond to saturating addition and subtraction. The state diagram is depicted in Figure 2(a).

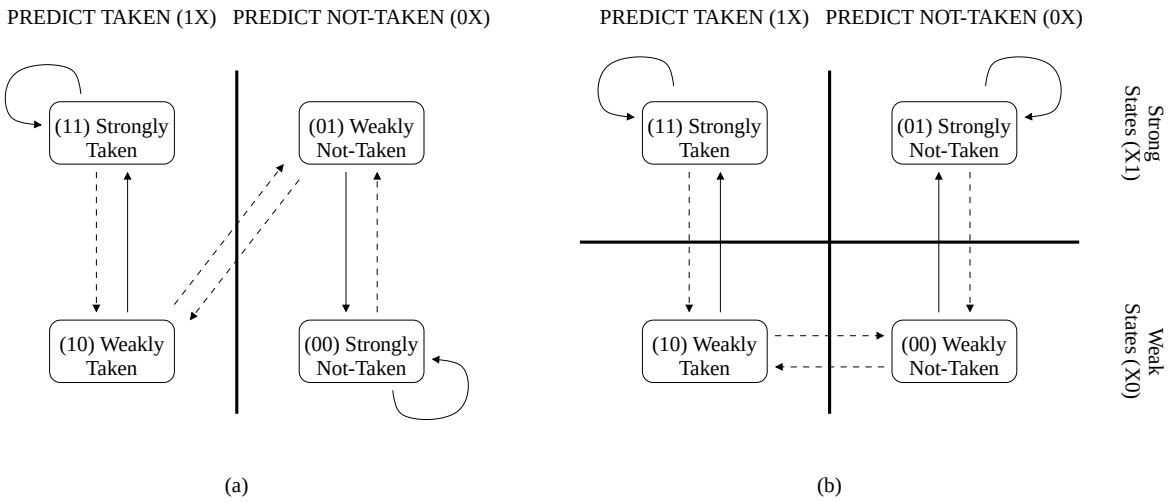

Figure 2: (a) The saturating 2-bit counter finite state machine (FSM). The most significant bit of the state encoding specifies the next predicted branch direction. (b) Another FSM that is functionally equivalent to the saturating 2-bit counter, but the states are renamed such that the most significant bit specifies the next prediction, and the least significant bit indicates a strong or weak prediction.

Solid arrows correspond to transitions made when the prediction was correct and dashed arrows correspond to the state transitions when there was a misprediction. The most significant bit of the state encoding is used to determine the direction of the branch prediction. For example, all states with an encoding of 1X (X denotes either 0 or 1) predict taken. By itself, the least significant bit does not convey any useful information. Paired with the direction bit, the least significant bit denotes a *strong* prediction when it is equal to the direction bit (states 00 and 11), and a *weak* prediction otherwise. This additional bit provides hysteresis so the branch predictor requires two successive mispredictions to change the predicted direction.

The assignment of states in a finite state machine are more or less arbitrary since the assigned states are merely names or labels. Because of this, an alternate encoding can be given to the two-bit counter. Figure 2(b) shows the state diagram for the renamed finite state machine. The state diagrams are isomorphic; only the labels for the two not-taken states have been exchanged. The most significant bit of the counter still denotes predicted direction, but the least significant bit can now be directly interpreted as being weak or strong. For example, if this hysteresis bit is 1, then a strong prediction was made; we refer to these states as the *strong states*. We call this renamed finite state machine the *split counter* because the strength of the state can be directly inferred from the hysteresis bit even when the hysteresis bit has been separated or *split* from the direction bit.

3.2 Bias of Counter States

The counters used in the agree predictor are effective because branches that alias to the same counter frequently agree with their separate biasing bits. This would make the agree counters tend to the

Benchmark Name	Data Set	Strong State Predictions
164.gzip	ref-graphic	0.900425
	ref-log	0.925201
	ref-program	0.890077
	ref-random	0.919332
	ref-source	0.901446
175.vpr	ref-place	0.860060
	ref-route	0.941841
176.gcc	ref-166	0.971020
	ref-200	0.937520
	ref-expr	0.953444
	ref-integrate	0.961725
	ref-scilab	0.937154
181.mcf	ref	0.915599
186.crafty	ref	0.921647
197.parser	ref	0.939315
243.perlbmk	ref-makerand	0.997504
	ref-splitmail	0.975547
	train-scrabble	0.959799
252.eon	ref-cook	0.969624
	ref-kajiya	0.927652
	ref-rushmeier	0.952164
254.gap	ref	0.946384
255.vortex	ref-1	0.974275
	ref-2	0.974035
	ref-3	0.975125
256.bzip2	ref-graphic	0.951196
	ref-program	0.930332
	ref-source	0.909522
300.twolf	ref	0.860225
Average		0.937213

Table 1: The fraction of strong state predictions is calculated by taking the number of branch predictions made in a strong state (strongly taken, or strongly not-taken), and dividing by the total number of predictions.

"agree strongly" state despite the aliasing. If the states of the regular two-bit counters tend to be heavily biased toward the strong states, then perhaps the bit used to provide hysteresis can also be shared among different branches.

To determine whether or not the hysteresis bits tend to be highly biased toward the strong states, we simulated a gshare predictor with 8192 (8K) two-bit counters in the pattern history table (PHT) and 13 bits of global branch history. We simulated one billion branches from each of the SPEC2000 integer benchmarks. Section 3.3.1 provides the full details of our simulation methodology. For each benchmark, we tracked the number of strong state predictions and the total number of predictions made. Table 1 shows how many of the dynamic branch predictions made in the SPEC2000 integer benchmarks were either strongly taken or strongly not-taken. Note that these statistics were collected for a small predictor which would have more interference than larger configurations.

For most benchmarks, the branch predictor counters remain in one of the two strong states for over 90% of the predictions made. Since most branches tend to be highly biased toward the strong states, this suggests that perhaps one bit per counter for hysteresis may be an overkill. Two questions naturally follow. First, how many hysteresis bits are actually needed per counter? Second, because the number of hysteresis bits needed per counter will be less than one, how can a "fractional-bit" counter be implemented? We answer the first question by obtaining an upper bound for the amount of information conveyed by the hysteresis bit. We then present the design of a counter that has a fractional-bit cost in Section 4.

3.3 Entropy of Hysteresis Bits

The hysteresis bit bias results suggest that it is very likely that the entropy of the bits is less than one bit per prediction. Let $S(x)$ be the *strength* of a counter x, where

$$S(x) = \begin{cases} 0 & \text{if } x \in \{\text{weakly taken, weakly not-taken}\} \\ 1 & \text{if } x \in \{\text{strongly taken, strongly not-taken}\} \end{cases}$$

which directly encodes whether the state of a counter is in a strong state or a weak state. Note that the value of the hysteresis bit of our split counter is always equal to S. To proceed, we need a way to measure the entropy of $S(x)$. Shannon's Theorem states that if the information rate of a source is less than or equal to the capacity of the channel, then there exists an encoding of the source symbols which allow the transmission of the symbols with arbitrarily small probability of errors even if the channel itself is noisy. For our purposes, we are not concerned about transmission errors. Our information is the strength of a predictor state $S(x)$, and our "channel capacity" is the number of bits we use to store this information (that is one bit per counter).

Our approach is to find a compact encoding of the symbols, which implies that the entropy of the source is not greater than the average bits per symbol in our compacted representation. For example, if we can compress 100 bits in a lossless manner down to 47 bits, then the entropy corresponding to the source that generated these bits is at most 0.47 bits per symbol. The entropy may be less if our encoding is not optimal, but compressing the symbols proves by construction that the entropy is bounded by the achieved compression rate. In this section, we do not provide an exact measure of the hysteresis bit entropy, but rather use data-compression techniques to provide an upper bound on $H(S)$.

3.3.1 METHODOLOGY

We used a simulation-based approach to generate all of the results presented in this paper. Our simulator is derived from the SimpleScalar toolset [15] [16]. We used the in-order simulator `sim-safe` to collect traces of one billion conditional branches from each of the simulated benchmarks. Due to the in-order nature of the branch trace generation, these traces do not include discarded branches from mispredicted execution paths. We use the traces to drive a branch predictor simulator to measure branch prediction accuracies.

We collected the branch traces from the SPEC2000 integer benchmarks. The benchmarks were compiled on an Alpha 21264 with `cc -g3 -fast -O4 -arch ev6 -non_shared`. The benchmarks, input sets and number of initial instructions skipped are all listed in Table 2. We collected exactly one billion conditional branches from each benchmark-input pair, except for `perlbmk.makerand`, `perlbmk.splitmail` and `vpr.place` which completed execution before one billion branches. Note that one billion branches approximately corresponds to five to six billion total instructions. The fastforward amounts were chosen to skip over the initial setup phases of the applications. When applicable, the sampling window was also placed to collect branches from different sections of the program [17]. For example, in the compression applications (`bzip2` and `gzip`), the branches are taken from both the compressing and decompressing phases of execution.

To measure the entropy conveyed by the strengths of predictor states, we instrumented our branch predictor simulator to generate a log of the values of $S(x)$. For each counter x_i in our 8K-entry gshare predictor, we maintained a separate trace for $S(x_i)$. At the end of each simulation, this provides one billion bits of data since each prediction provides one symbol. We concatenated

Benchmark Name	Input Set	Instructions Skipped ($\times 10^9$)
164.gzip	ref-graphic	37.95
	ref-log	15.55
	ref-program	24.425
	ref-random	30.89
	ref-source	30.375
175.vpr	ref-place	0.0
	ref-route	24.975
176.gcc	ref-166	7.85
	ref-200	19.5
	ref-expr	4.175
	ref-integrate	2.5
	ref-scilab	9.9
181.mcf	ref	20.675
186.crafty	ref	44.3
197.parser	ref	18.4
243.perlbmk	ref-makerand	1.025
	ref-splitmail	0.0
	train-scrabble	22.025
252.eon	ref-cook	11.5
	ref-kajiya	16.5
	ref-rushmeier	8.0
254.gap	ref	29.5
255.vortex	ref-1	26.7
	ref-2	13.1
	ref-3	14.85
256.bzip2	ref-graphic	49.7
	ref-program	45.4
	ref-source	42.6
300.twolf	ref	10.125

Table 2: The SPEC2000 integer benchmarks and data sets used in our simulations, along with the number of initial instructions skipped before collecting a trace of one billion conditional branches.

all of the individual traces of $S(x_i)$ into a single file and then encoded the data with the `gzip` compression program using the `--best` flag. The size of the compressed trace divided by the size of the original trace provides an upper bound on the entropy of $S(x)$. Using more advanced compression techniques such as arithmetic coding could yield tighter bounds on the entropy, but our results from using `gzip` are sufficient to motivate our design of a more efficient predictor organization.

3.3.2 RESULTS

The strong bias toward high confidence branch predictor states suggests that the information content of the counter strength S is less than one bit per prediction. The results in Table 3 demonstrate that this is indeed the case. For each benchmark and data set, Table 3 lists the size of the trace of the samples of S, the compressed size of the trace when using `gzip`, and the bound on the entropy of S. There are no benchmarks that ever use more than one half bit per prediction, and on average only about 0.2 bits are needed.

Information theory states that with the correct encoding, we could in fact design a branch predictor counter that only uses 0.44 bits per counter for hysteresis (`vpr.place` has the highest entropy) and still perform exactly the same as a full two-bit counter. The problem is that the process of determining an optimal code as well as the corresponding encoding/decoding logic are almost certainly too expensive to implement in hardware. Instead, we propose a much simpler scheme that uses fewer total hysteresis bits at the cost of a slight degradation in prediction accuracy.

Benchmark Name	Input Set	Original Trace Size (MB)	Compressed Size (MB)	Entropy (bits/symbol)
164.gzip	ref-graphic	119.213	38.060	0.319261
	ref-log	119.213	29.356	0.246252
	ref-program	119.213	40.739	0.341731
	ref-random	119.213	29.139	0.244429
	ref-source	119.213	39.771	0.333611
175.vpr	ref-place	35.760	15.549	0.434809
	ref-route	119.213	19.435	0.163027
176.gcc	ref-166	119.213	8.355	0.070087
	ref-200	119.213	23.907	0.200541
	ref-expr	119.213	16.099	0.135042
	ref-integrate	119.213	8.065	0.067652
	ref-scilab	119.213	23.583	0.197822
181.mcf	ref	119.213	24.069	0.201896
186.crafty	ref	119.213	38.190	0.320349
197.parser	ref	119.213	23.690	0.198723
243.perlbmk	ref-makerand	21.440	0.146	0.006794
	ref-splitmail	9.883	0.828	0.083780
	train-scrabble	119.213	2.458	0.020623
252.eon	ref-cook	119.213	1.855	0.015564
	ref-kajiya	119.213	26.265	0.220325
	ref-rushmeier	119.213	11.800	0.098980
254.gap	ref	119.213	7.627	0.063975
255.vortex	ref-1	119.213	3.210	0.026925
	ref-2	119.213	2.723	0.022838
	ref-3	119.213	3.230	0.027091
256.bzip2	ref-graphic	119.213	25.485	0.213781
	ref-program	119.213	29.562	0.247976
	ref-source	119.213	36.499	0.306168
300.twolf	ref	119.213	49.465	0.414935
Average				0.182895

Table 3: The entropy for the hysteresis bits is bounded by compressing a trace of all of the hysteresis bits.

4. Shared Split Counters

Our entropy results from the previous section suggest that it is possible to design branch predictor structures that use less storage than traditional two-bit counters while maintaining comparable prediction accuracies. We propose using *shared split counters* (SSC) which reduce the storage requirements of two-bit counter based predictors. This section first describes the hardware organization of shared split counter arrays, and then evaluates the performance of SSC branch predictors.

4.1 Counter Organization

We first discuss the hardware organization of our shared split counter predictors. The first two aspects of the SSC predictors are array separation and shared hysteresis bits, both of which have apparently been known and used in industry, but have only recently been published by the EV8 design team [14]. The last component of our SSC predictors is the split counter finite state machine described in Section 3 that is similar to a saturating two-bit counter, but the hysteresis bit directly encodes the value of the strength of the counter.

4.1.1 ARRAY SEPARATION

In a conventional branch predictor counter array organization, the prediction counters are stored in a SRAM structure. Each entry of the SRAM consists of one two-bit counter, as illustrated in Figure 3a. To perform a prediction, the predictor logic generates an index into the array. This index may simply be the least significant bits of the branch address, or it may also include other information such as branch history. The logic uses the direction bit of the indexed counter, and predicts taken if the bit is one, and not-taken if the bit is zero. At the time of the predictor update, the finite state machine logic uses the state encoded by both the direction and hysteresis bits and the actual branch outcome to update the counter state.

An important observation is that the hysteresis bit is never directly needed for the lookup phase of the branch predictor. As a result, the direction bits and the hysteresis bits may be stored in two physically separate SRAM arrays, as shown in Figure 3b. Although the bits from two corresponding entries in the direction array and hysteresis array still form a single logical two-bit counter, partitioning the array may allow each individual SRAM to be smaller, and therefore faster. Since the update phase of the predictor is not on the critical path, the hysteresis array may actually be placed in an entirely different location from the direction array.

4.1.2 SHARED HYSTERESIS BITS

With a single monolithic array of n two-bit counters, it is only natural that there is a total of n direction bits and n hysteresis bits. When the direction and hysteresis state is separated into two different arrays, there is no reason that these two arrays must have the same number of entries. In the EV8 branch predictor, the number of entries in the hysteresis array is half that of the direction array. This means that every two direction array bits share a single hysteresis bit. Figure 4a illustrates the indexing of an eight-entry direction array and a four-entry hysteresis array with the same branch address. Address A maps to entry 110_2 in the direction array. In the case of the hysteresis array, the index function ignores one additional address bit, mapping to entry 10_2. Figure 4b illustrates conventional interference where a different address B maps to the exact same direction and hysteresis entries as address A. With shared counters, a third form of interference may occur. Figure 4c

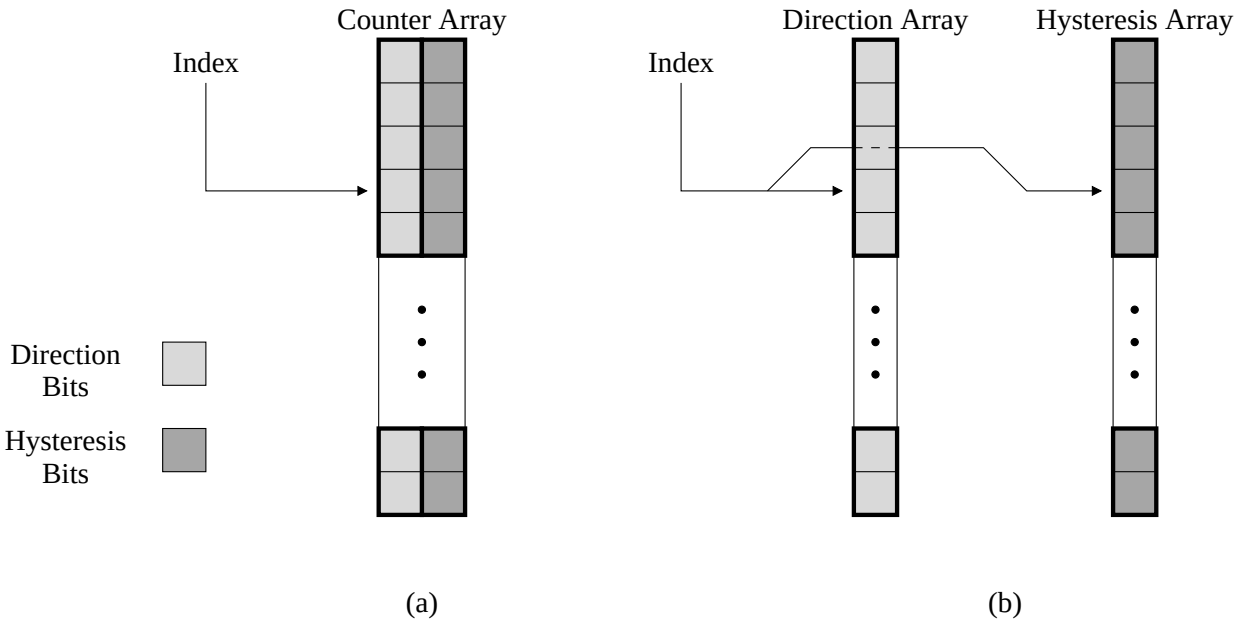

Figure 3: (a) A single SRAM structure stores both direction and hysteresis bits for each counter. (b) Two SRAM structures store the direction and hysteresis bits in separate locations.

shows an address C that maps to an entry in the direction array that does not conflict with either A or B. On the other hand, there still remains an aliasing conflict in the hysteresis array. While shared hysteresis bits reduce the area requirements of the hysteresis array, this additional interference will cause additional branch mispredictions.

4.1.3 SPLIT COUNTERS

Using shared hysteresis bits may increase the amount of interference in the hysteresis array. Although most counters spend the majority of the time in one of the strong states, sharing hysteresis bits may make this impossible. Consider the two counters in Figure 5a where the counter addressed by A is in the strongly taken state (11_2) and counter addressed by B is in the strongly not-taken state (00_2). With the hysteresis bit sharing shown in Figure 5b, one of the two counters will be forced into a weak state because the hysteresis bit has different values for the ST and SNT states in a conventional 2bC encoding.

The split counter encoding of the two-bit counter provides a situation where the hysteresis bit is the same for both ST and SNT states. The result is that two counters that share hysteresis bits may both be in the strong states at the same time, as illustrated in Figure 5c.

In the remainder of this section, we consider *shared split counter* (SSC) branch predictors that use shared hysteresis arrays with the split counter FSM encoding. The notation SSCn gshare denotes a gshare predictor that uses the split counter encoding, and a hysteresis array that is n times smaller than the original. Note that a SSC-1 predictor behaves identically to the corresponding 2bC based predictor.

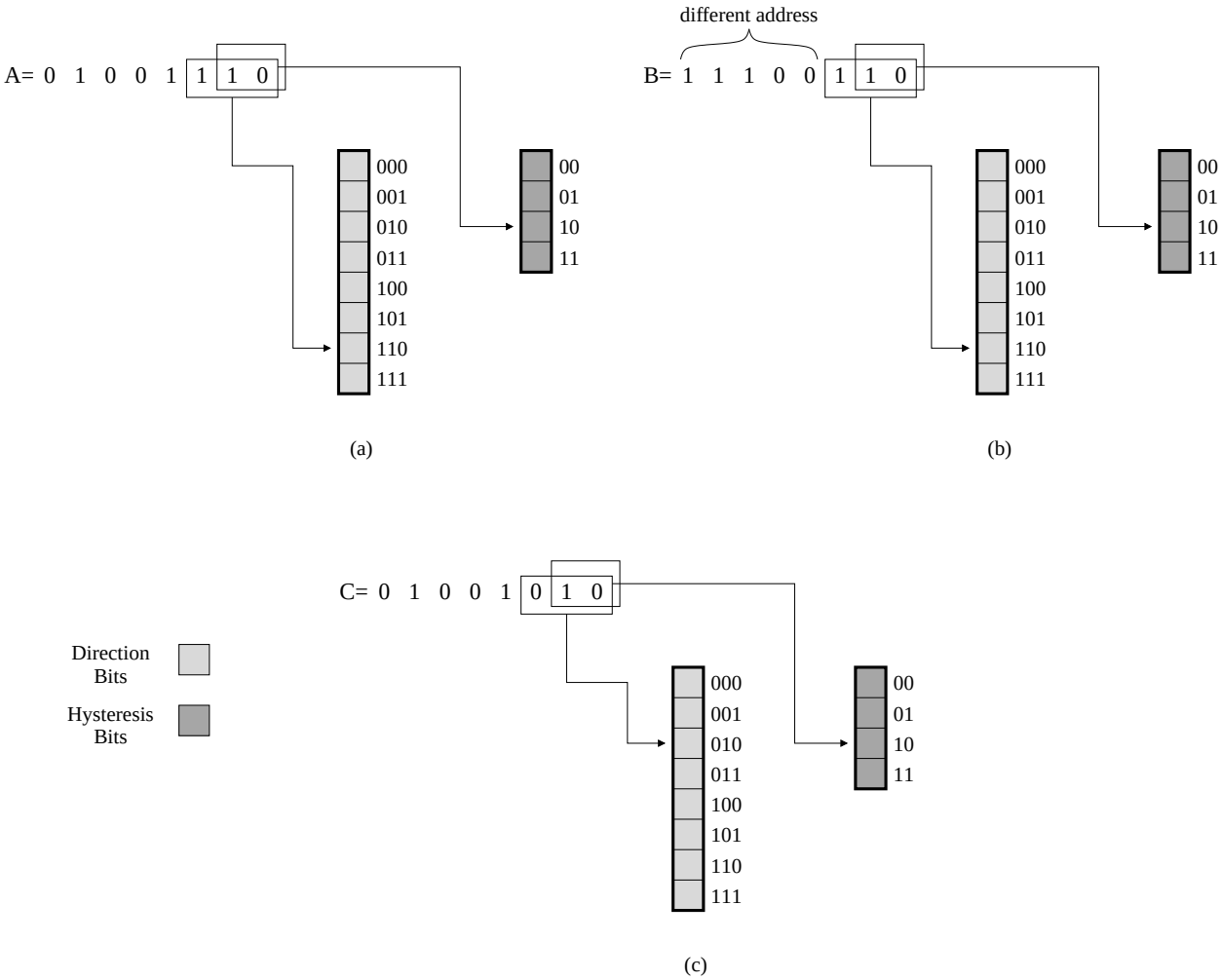

Figure 4: (a) Mapping the branch address A to different sized direction and hysteresis arrays. (b) The branch address B is different from A, but still maps to the same counter entries. (c) Aliasing only in the hysteresis array.

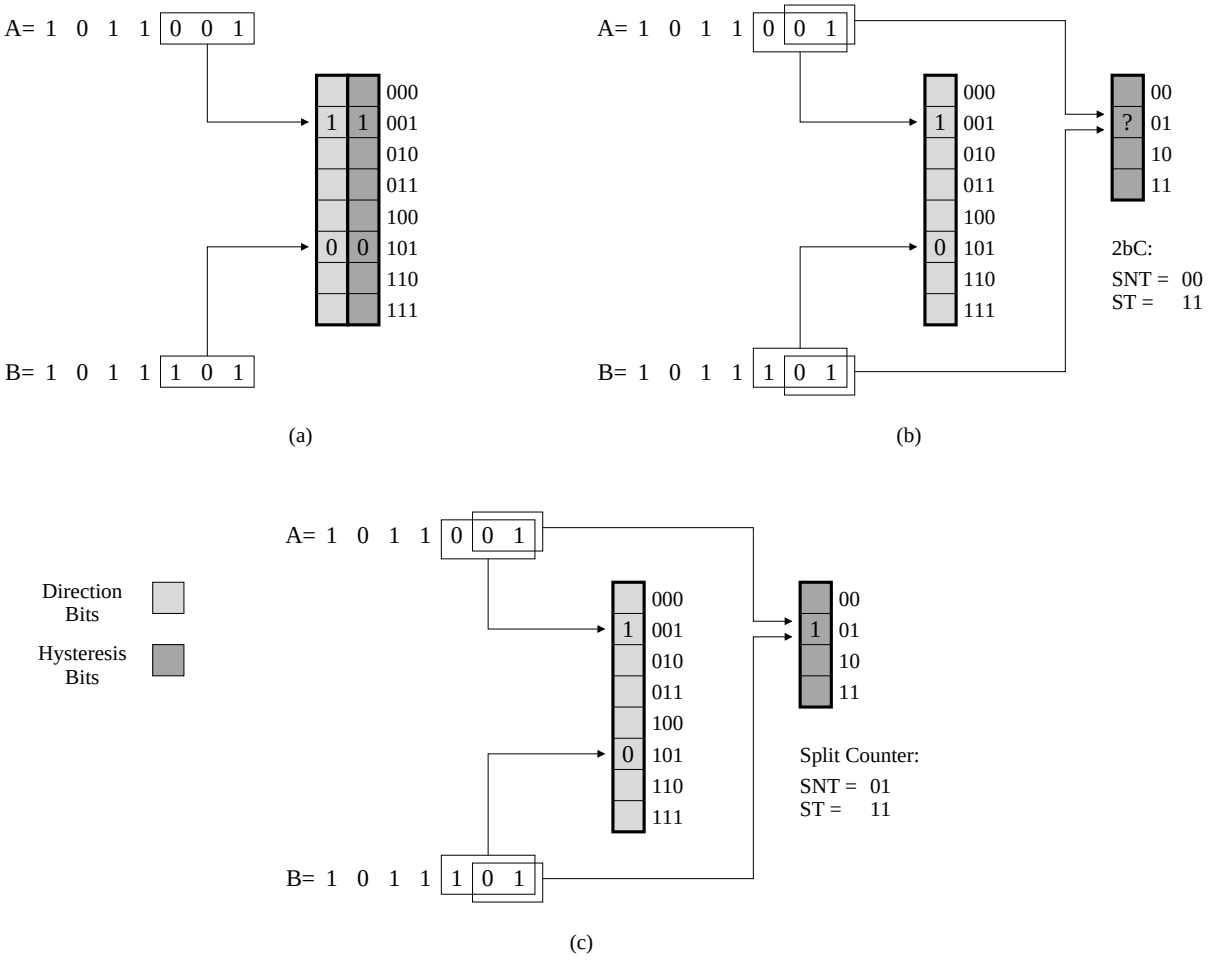

Figure 5: (a) Two distinct counters in a 2bC-based array may simultaneously be in strong states. (b) Sharing hysteresis bits without the split counter encoding prevents counters which alias in the hysteresis array to be in both strongly taken (ST) and strongly not-taken (SNT) states simultaneously. (c) The split counter encoding allows counters that share a hysteresis bit to both be in strong states.

4.2 Performance

In the remainder of this section, we present our simulation results to evaluate the performance impact of SSC predictors. In particular, we first focus on a SSC gshare predictor, and then also briefly explore the impact of using shared split counters on a more sophisticated 2bC-based predictor.

4.2.1 GSHARE PREDICTORS

We first examine SSC versions of the 8K entry gshare predictor from Section 3. Table 4 shows the misprediction rates for three predictors. The first is the original 2bC gshare, the second is for a SSC2 gshare predictor, and the third is a SSC4 gshare predictor. Not surprisingly, as the degree of sharing increases, the misprediction rates of the SSC predictors also increase. The bottom of Table 4 also lists the misprediction rate differential, which is the relative increase in the misprediction rate. Going from a 2bC gshare to the SSC2 gshare increases the relative number of mispredictions by 2.8% on average, but reduces the SRAM storage requirements when measured by total number of bits by 25%. Further increasing the degree of sharing to the SSC4 gshare provides a smaller additional area savings of 12.5% for a total of 37.5% over the original gshare, while increasing mispredictions by another 5.5%.

The entropy of the hysteresis bit also increases with the degree of sharing, as shown in Table 5. When two counters are forced to share a single hysteresis bit, the entropy only increases slightly from 0.181 bits/prediction to 0.194 bits/prediction. Note that the area cost for the 2bC gshare is one full bit of hysteresis per counter, while SSC2 uses only 0.5 bits per counter. When the degree of sharing increases to four (SSC4), the entropy increases by a greater amount, and this is reflected by the increase in mispredictions. From an information theoretic perspective, using an SSC organization makes better use of the hysteresis bits because each bit effectively conveys more information and SSC uses fewer total bits.

The shared split counters achieve a greater coding efficiency. By coding efficiency, we mean the amount of information divided by the storage used. For example, the 2bC gshare uses a full hysteresis bit for every counter, but only provides 0.1809 bits per prediction. This has a coding efficiency of 18.1%. In the SSC2 gshare, a single hysteresis bit is shared between two logical counters, which can be viewed as 0.5 bits of hysteresis per counter. With an entropy of 0.1937 bits per prediction, the SSC2 gshare achieves a coding efficiency of 38.7%. The SSC4 gshare has a cost of 0.25 hysteresis bits per counter, and thus has a 84.4% coding efficiency. Going to a SSC8 predictor would exceed the "channel capacity" of the hysteresis bits, and we would expect very high misprediction rates. That is, eight hysteresis counters' worth of information can not be encoded with a single bit for any encoding[1]. We simulated SSC8 gshare predictors and found that such a high degree of sharing results in so many mispredictions that the next smaller sized 2bC gshare predictor is more accurate (see Figure 6).

From a practical point of view, it is still unclear whether a SSC gshare predictor provides a better area-performance tradeoff than a traditional 2bC gshare. To answer this, we simulated 2bC and SSC gshare predictors over a range of predictor sizes. Figure 6 shows the misprediction rates of a traditional gshare predictor along with SSC2 and SSC4 gshare predictors.[2]

1. Assuming our entropy bounds are not more than 59.2% over the true value of $H(S(x))$.
2. Figure 6 also includes the performance of a Bi-Mode predictor because the scale does not start at zero. We only include the Bi-Mode predictor to provide a reference for making more meaningful comparisons between the 2bC and SSC gshare predictors.

Benchmark (name.input)	Misprediction Rate		
	2bC gshare	SSC2 gshare	SSC4 gshare
bzip2.graphic	0.0488	0.0500	0.0508
bzip2.program	0.0697	0.0719	0.0720
bzip2.source	0.0905	0.0939	0.0957
crafty	0.0783	0.0808	0.0852
eon.cook	0.0304	0.0318	0.0379
eon.kajiya	0.0723	0.0741	0.0764
eon.rushmeier	0.0478	0.0500	0.0556
gap	0.0536	0.0563	0.0594
gcc.166	0.0290	0.0301	0.0319
gcc.200	0.0625	0.0649	0.0685
gcc.expr	0.0465	0.0483	0.0511
gcc.integrate	0.0383	0.0398	0.0419
gcc.scilab	0.0628	0.0657	0.0698
gzip.graphic	0.0996	0.1005	0.1017
gzip.log	0.0748	0.0755	0.0763
gzip.program	0.1099	0.1108	0.1118
gzip.random	0.0807	0.0809	0.0814
gzip.source	0.0985	0.0992	0.1001
mcf	0.0844	0.0854	0.0875
parser	0.0607	0.0617	0.0637
perlbmk.makerand	0.0025	0.0039	0.0193
perlbmk.splitmail	0.0244	0.0261	0.0277
perlbmk.scrabble	0.0402	0.0444	0.0551
twolf	0.1398	0.1423	0.1451
vortex.1	0.0257	0.0281	0.0343
vortex.2	0.0260	0.0280	0.0347
vortex.3	0.0249	0.0270	0.0341
vpr.place	0.1399	0.1418	0.1444
vpr.route	0.0582	0.0584	0.0589
Average	0.0628	0.0645	0.0680
Relative Mispred. Increase	+0.0%	+2.8%	+8.3%
SRAM Storage	-0.0%	-25.0%	-37.5%

Table 4: The misprediction rates for the SPEC2000 benchmarks for 2bC and SSC versions of an 8K-entry gshare predictor. The relative increase in the misprediction rate and the decrease in SRAM storage requirements are with respect to the 2bC gshare configuration.

Benchmark (name.input)	Entropy (bits/prediction)		
	2bC gshare	SSC2 gshare	SSC4 gshare
bzip2.graphic	0.2138	0.2205	0.2280
bzip2.program	0.2480	0.2588	0.2682
bzip2.source	0.3062	0.3205	0.3350
crafty	0.3203	0.3516	0.3894
eon.cook	0.0156	0.0176	0.0189
eon.kajiya	0.2203	0.2430	0.2763
eon.rushmeier	0.0990	0.1059	0.1210
gap	0.0640	0.0742	0.0934
gcc.166	0.0701	0.0796	0.0917
gcc.200	0.2005	0.2238	0.2522
gcc.expr	0.1350	0.1516	0.1741
gcc.integrate	0.0677	0.0782	0.0921
gcc.scilab	0.1978	0.2218	0.2527
gzip.graphic	0.3193	0.3311	0.3553
gzip.log	0.2463	0.2568	0.2712
gzip.program	0.3417	0.3614	0.3774
gzip.random	0.2444	0.2477	0.2719
gzip.source	0.3336	0.3495	0.3627
mcf	0.2019	0.2112	0.2272
parser	0.1987	0.2085	0.2237
perlbmk.makerand	0.0068	0.0077	0.0080
perlbmk.splitmail	0.0838	0.0892	0.0988
perlbmk.scrabble	0.0206	0.0239	0.0322
twolf	0.4149	0.4544	0.5022
vortex.1	0.0269	0.0336	0.0442
vortex.2	0.0228	0.0288	0.0388
vortex.3	0.0271	0.0337	0.0443
vpr.place	0.4348	0.4655	0.4997
vpr.route	0.1630	0.1668	0.1709
Average	0.1809	0.1937	0.2111

Table 5: The upper bound on the hysteresis bit entropy as determined by compression of the hysteresis traces.

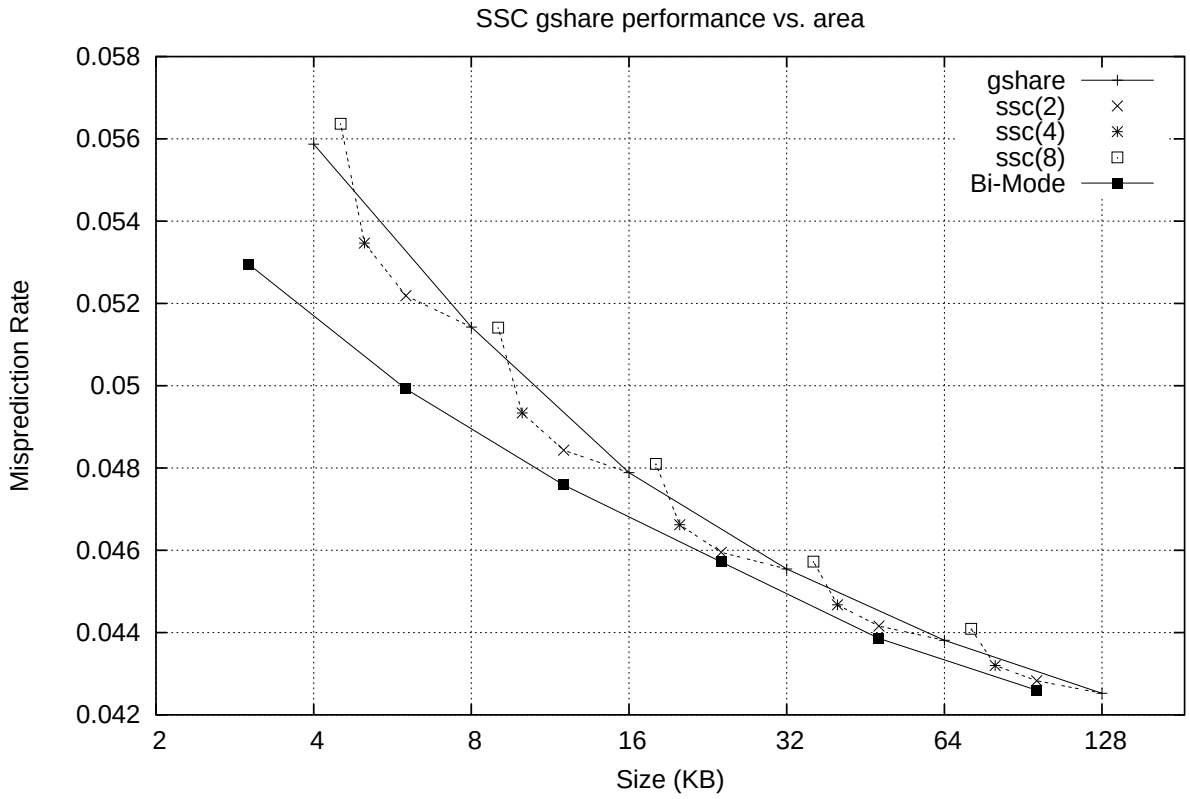

Figure 6: The misprediction rates for 2bC and SSC gshare predictors. The Bi-Mode predictor performance is provided for reference because the y-axis does not start at zero.

Although an 8K entry SSC gshare makes more mispredictions than a regular 2bC gshare, the SSC predictors achieve superior performance per area. The dashed lines in Figure 6 connect configurations with the same number of direction bits, but different amounts of hysteresis bits. A flatter dashed line indicates that there is less accuracy degradation for the savings in area. For example, the SSC2 versions of the 32KB, 64KB and 128KB gshare all suffer less than a 0.9% relative increase in mispredictions (an increase in the absolute misprediction rate by 0.00041). Also note that the x-axis is on a logarithmic scale, and so the area savings are greater than they might otherwise appear. At lower hardware budgets, the effects of interference decrease the effectiveness of shared split counters.

4.2.2 BI-MODE PREDICTORS

Many branch prediction algorithms use saturating two-bit counters. To demonstrate the applicability of shared split counters, we present the performance results for a Bi-Mode predictor implemented with shared split counters. The Bi-Mode predictor consists of two direction PHTs and a choice PHT, all three of which use saturating two-bit counters, as shown in Figure 7. The exclusive-or of b

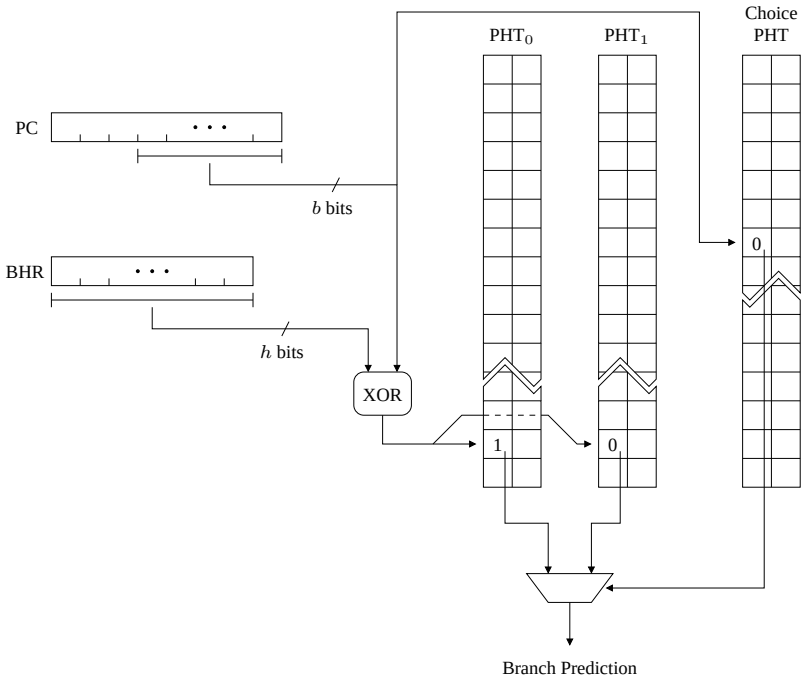

Figure 7: The Bi-Mode predictor has three separate PHTs, each of which may use shared split counters.

branch address bits and h history bits index into the direction PHTs as in the gshare predictor. The b branch address bits also index into the choice PHT.

As shown in Figure 6, the Bi-Mode predictor achieves superior performance per area when compared to any of the SSC gshare configurations. Directly comparing the Bi-Mode predictor to an SSC gshare predictor is not entirely fair because the two approaches address different issues in the branch predictors (again, we only included the Bi-Mode in Figure 6 as a reference point). The Bi-Mode predictor attacks the problem of PHT interference by splitting branches into a taken-biased substream and a not-taken biased substream. On the other hand, the SSC gshare addresses the fact that the hysteresis bits of the PHT are underutilized, and provides a means of trading a little accuracy for space reduction. These issues are largely orthogonal and applying shared split counters to a Bi-Mode predictor (or any other two-bit counter based predictor such as gskewed or YAGS) should further improve the performance-versus-area curve.

We simulated several configurations of SSC Bi-Mode predictors and Figure 8 shows the results. A SSC Bi-Mode configuration labeled as Bi-Mode(m, n) uses a 1:m reduction of hysteresis bits in the choice PHT, and a 1:n reduction of hysteresis bits in both of the two direction PHTs. Similar to the SSC gshare predictor, the shared split counters are not as effective at smaller hardware budgets. We can view the 2bC Bi-Mode predictor (for configurations larger than 8KB) as a way to reduce the SRAM requirements of a 2bC gshare predictor by 33% while maintaining approximately the same misprediction rate. On the other hand, the SSC Bi-Mode(4,2) configuration provides a 29% space reduction over the 2bC Bi-Mode, or a 47% reduction over the original 2bC gshare.

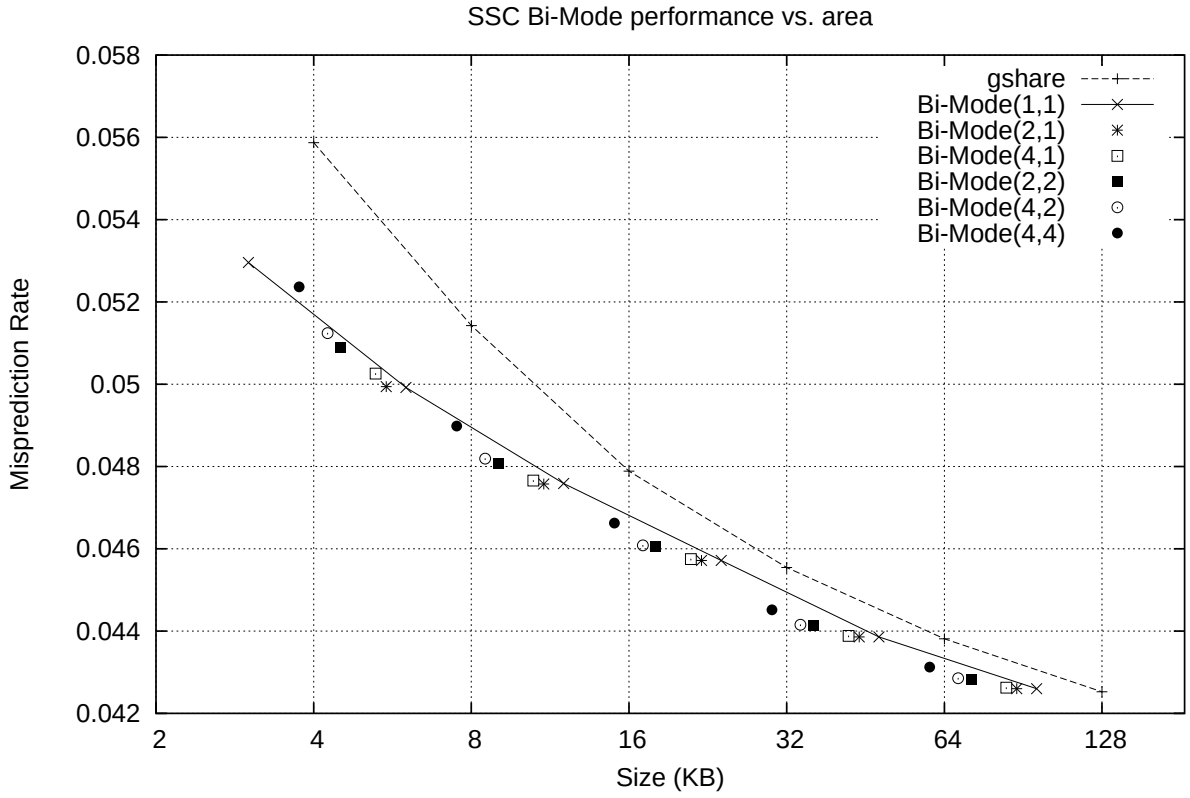

Figure 8: The misprediction rates for 2bC and SSC Bi-Mode predictors. A Bi-Mode(m,n) predictor has $1/m$ as many hysteresis bits in the choice PHT and $1/n$ as many hysteresis bits in the direction PHTs.

5. Why Shared Split Counters Work

We have shown that the shared split counter gshare predictor provides a better performance-area tradeoff than a traditional saturating two-bit counter gshare. In this section, we analyze the behavior of the hysteresis bit in the SSC2 gshare predictor to explain why the sharing of state between counters does not greatly affect prediction accuracy. In particular, we illustrate how two pattern history table entries sharing a single hysteresis bit can transition between states without interfering with its neighboring entry, qualitatively describe the new sources of interference that shared split counters may introduce, and then quantitatively measure the frequency of these situations and their impact on overall prediction accuracy.

5.1 When SSC Works

Assume the processor executes the following piece of code, and the branches marked A and B map to pattern history table entries that share a hysteresis bit.

```
i=0;
do
{
        for (j=0; j<50; j++)
        {
                if ( i % 2 )            B
                        ...
        }
        i++;
}
while (i<1000);          A
```

For each iteration of the do-while loop, the inner branch marked B will alternate between always not-taken and always taken. Figure 9 shows the state for the two neighboring pattern history table entries that share a single hysteresis bit. In this example, A is initially predicted strongly taken while B is initially predicted strongly not-taken. Now suppose that the program has just completed one iteration of the while loop and is entering the next. For the next 50 instances, branch B will be taken. The next prediction (prediction 1 in Figure 9) for B will not be correct, causing the shared hysteresis bit to enter the weak state. Another misprediction (prediction 2) causes the direction bit for B to change to taken. Finally, the prediction for branch B will be correct (prediction 3), causing the shared hysteresis bit to return to a strong state. At this point, the state that branch A "sees" is the same as before. Since branch A is not executed while branch B transitions from one strong state to the other, it is not affected by the transition.

5.2 When SSC Introduces Interference

The sharing of hysteresis bits between counters can lead to situations where a branch affects the state (and hence predictions) of an otherwise unrelated branch. In this section, we describe the possible scenarios for *sharing-induced* interference.

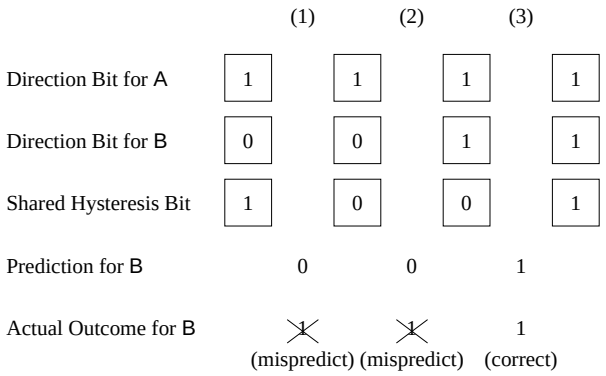

Figure 9: A counter can switch from not-taken to taken (or vice-versa) without interfering with the counter that also shares the same hysteresis bit.

We categorize branch mispredictions into five possible categories. The list below briefly summarizes the five cases, and a more detailed explanation follows.

1. **Normal** mispredictions occur when a branch direction changes and two mispredictions occur before the predicted direction also changes. This corresponds to the normal behavior of a saturating two-bit counter.

2. **Transient** mispredictions occur when the branch prediction does not change, but a single misprediction is observed. Transient mispredictions normally occur in saturating two-bit counters.

3. **Weak Hysteresis** mispredictions occur when a branch direction has changed, but only a single misprediction is observed before the predicted direction also changes.

4. **Dueling Counter** mispredictions occur when a branch direction has changed, but the corresponding counter experiences more than two mispredictions due to interference in the hysteresis bit.

5. **Transient Dueling Counter** mispredictions occur when a branch direction does not change, but the corresponding counter experiences two or more mispredictions due to interference in the hysteresis bit.

5.2.1 NORMAL MISPREDICTIONS

Normal mispredictions correspond to the typical behavior of saturating two-bit counters. Figure 10 shows a series of branch outcomes corresponding to a single two-bit counter. When the branch outcomes change from taken to not-taken, two mispredictions occur before the predicted direction also changes. We classify these two mispredictions as *normal*.

5.2.2 TRANSIENT MISPREDICTIONS

Transient mispredictions also occur in conventional two-bit counters. Figure 11 shows a series of branch outcomes corresponding to a loop termination branch. Typically the branch is taken, but

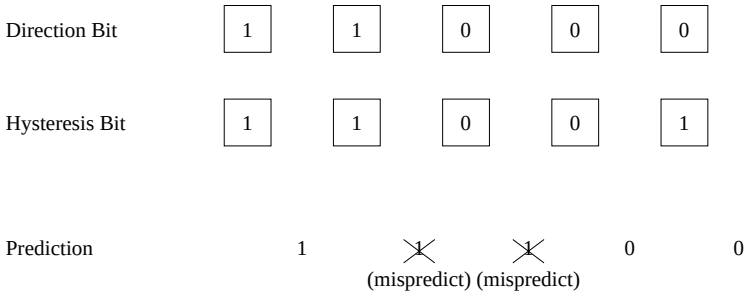

Figure 10: Normal mispredictions occur when the branch direction changes and the counter experiences exactly two mispredictions.

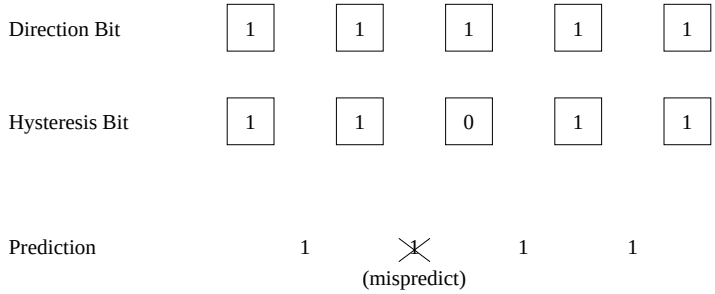

Figure 11: A transient misprediction occurs when there is a single misprediction that does not change the counter's predicted direction.

there is a single instance when the loop is not-taken for the branch exit. This situation was the original motivation for using two bits instead of a single bit. This single branch misprediction is classified as *transient*.

5.2.3 WEAK HYSTERESIS MISPREDICTIONS

Multiple counters that share a single hysteresis bit may interfere with each other. This may lead to situations where additional mispredictions occur that otherwise would not have if full saturating two-bit counters were used. Note that while branch B (from the example of Section 5.1) is making the transition from the strongly not-taken state to the strongly taken state, the state for branch A is temporarily changed from strongly taken to weakly taken and then back. If branch A were to mispredict during this interval, the direction bit would immediately change to the not-taken direction. In this situation, the shared hysteresis bit does not provide any hysteresis, allowing the direction bit for branch A to change after only a single misprediction. We classify all mispredictions that cause a change in the stored direction after exactly one misprediction as *weak hysteresis* mispredictions.

5.2.4 DUELING COUNTER MISPREDICTIONS

The next case for introducing interference when using a shared hysteresis bit is the case of *dueling counters*. Figure 12 illustrates the dueling counter scenario. The counter Y is originally in a strong

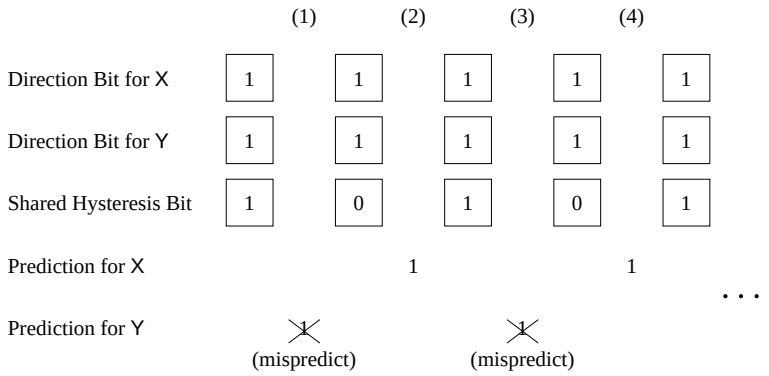

Figure 12: Two alternating predictions to entries sharing a confidence bit can result in one of the entries not being able to switch its prediction bit.

state, but each time Y mispredicts, the hysteresis bit is set to zero. But if counter X makes a correct prediction before Y's next prediction, then this returns the hysteresis bit to one, thus forcing Y back into a strong state when it would not be if the hysteresis bit was not shared.

A sequence of dueling counters can result in two possible outcomes. The first is that the correctly predicted branch, X in Figure 12, stops interfering and allows the mispredicting branch Y to change its direction bit. These mispredictions are classified as *dueling counter mispredictions*. The second possible outcome is that the behavior of branch Y changes such that the branch no longer mispredicts and the counter returns to a strong state. We classify these mispredictions as *transient dueling* because they do not result in a change in the counter's direction bit.

5.3 Classifying Mispredictions

Classifying mispredictions requires only tracking the number of consecutive mispredictions for each counter, and whether the predicted direction changed on the next correct prediction. Table 6 shows the method for classifying mispredictions. Normal mispredictions occur when the number of mispredictions is exactly two, and the direction changes. Weak hysteresis mispredictions occur when the number of mispredictions is exactly one, and the direction changes. Dueling counter mispredictions occur when the number of consecutive mispredictions is greater than two, and the direction changes. Transient and transient dueling mispredictions cover all remaining cases where one or more mispredictions occur, but the final direction is the same as before the first misprediction. From Table 6, it is easy to see that any misprediction can be classified, and that the classification will be unique.

5.4 Results

Table 7 lists the counts of each class of mispredictions for a few benchmarks and the average for the 8K entry PHT configurations of a regular gshare, an SSC2-gshare and an SSC4-gshare. The benchmark `gcc.166` represents a case that has approximately average behavior, `gzip.random` is a case where shared split counters causes a very small relative increase in the branch misprediction

Class	Pattern	Number of Mispredictions	Branch Direction Changed?
Normal	1 1 1 **0 0** 0 0 0 0 0 0 0 **1 1** 1 1 1 1	2	yes
Weak Hysteresis	1 1 1 **0** 0 0 0 0 0 0 0 0 **1** 1 1 1 1 1	1	yes
Dueling Counters	1 1 1 **0 0 ... 0** 0 0 0 0 0 **1 1 ... 1** 1 1	≥ 3	yes
Transient	1 1 1 **0** 1 1 1 1 0 0 0 **1** 0 0 0 0	1	no
Transient Duel	1 1 1 **0 ... 0** 1 1 1 0 0 0 **1 ... 1** 0 0 0	≥ 2	no

1 = Taken Branch
0 = Not-Taken Branch
Bold = misprediction

Table 6: The five possible cases in our misprediction classification scheme.

rate, and `vortex.1` is a case that results in a larger relative increase in the branch misprediction rate.

When measuring dueling mispredictions, we count the first two mispredictions as normal mispredictions, and only the third misprediction and beyond as dueling mispredictions. This is due to the fact that with regular two bit counters, we would still observe two mispredictions before the predicted branch direction changed. The average duel length (number of mispredictions per duel) does not include the initial two mispredictions either. The reason for not including the initial two mispredictions for the dueling counter statistics is to emphasize the *additional* interference in the direction bits that the sharing counters introduce. Similarly for transient dueling mispredictions, the first misprediction is not counted toward the run length. The full misprediction classification results for all benchmarks is located in Appendix A.

The results from Table 7 show that the amount of shared counter-induced interference is not too large. The trends across all benchmarks are very similar. The additional interference due to the shared split counters causes additional mispredictions due to dueling counter and transient dueling mispredictions. This increase of sharing induced interference is partially offset by a reduction in the number of weak hysteresis and normal mispredictions. The overall effect is that for the same number of PHT entries, the shared split counters increase the branch misprediction rate slightly, but makes up for it by reducing the overall storage requirements. Across benchmarks, the relative increase in the misprediction rates is strongly correlated with the lengths of dueling counter runs.

6. Conclusions

We presented a different encoding of the traditional saturating two-bit counter where the least significant bit explicitly represents whether the current state is strong or weak, while the most significant bit still represents the predicted branch direction. The separation of the hysteresis information al-

	gshare	SSC2	SSC4
gcc.166			
Normal	10784448	9952886	8846986
Transient	14064655	13927004	13714395
Weak	4123134	3817670	3484023
Dueling	0	628143	1389720
Avg. Duel Length	-	2.445	2.748
Transient Dueling	0	1747995	4408238
Avg. Duel Length	-	2.509	2.820
Mispred. Rate	0.0290	0.0301	0.0318
Entropy	0.0701	0.0796	0.0917
gzip.random			
Normal	31151328	31067716	29499944
Transient	33955660	33948868	34160162
Weak	15555084	15453692	13480777
Dueling	0	83441	883483
Avg. Duel Length	-	2.021	1.747
Transient Dueling	0	305790	3348454
Avg. Duel Length	-	2.908	1.689
Mispred. Rate	0.0807	0.0809	0.0814
Entropy	0.2444	0.2477	0.2719
vortex.1			
Normal	8202454	7981606	7417620
Transient	14103166	13971917	14291643
Weak	3411132	3252496	2187239
Dueling	0	562426	2401107
Avg. Duel Length	-	5.738	12.419
Transient Dueling	0	2082799	7010855
Avg. Duel Length	-	6.843	10.322
Mispred. Rate	0.0257	0.0281	0.0343
Entropy	0.0269	0.0336	0.0442
Average			
Normal	22194123	21140291	19738131
Transient	26664639	26480527	26218850
Weak	9803559	9272519	8766624
Dueling	0	802374	1980424
Transient Dueling	0	2244268	5635727
Mispred. Rate	0.0628	0.0645	0.0680
Entropy	0.1809	0.1937	0.2111

Table 7: Misprediction classification statistics for `gcc` (approximately average case), `gzip` (small amount of interference), `vortex` (large amount of interference) and the average over all of the benchmarks.

lows the counter to be separated into two distinct parts. Using information theoretical techniques, we empirically show that the least significant bit of saturating 2-bit counters conveys less than 0.18 bits per prediction. We proposed sharing a single hysteresis bit between multiple counters, such that separate bits are maintained for the predicted branch direction, although additional interference may occur in the shared hysteresis bit. This achieves a cost of less than 2 bits per counter. Using simulations, we show that replacing the 2-bit counters with shared split counters in the PHTs of the gshare and Bi-Mode predictors rates provides comparable prediction rates at significantly lower hardware costs.

The idea of using shared split counters can potentially be applied to other areas of branch predictor design. For instance the meta-predictor used in the tournament hybrid branch predictor [7] which consists of a table of saturating 2-bit counters is a candidate for using shared split counters, especially when combined with two sub-predictors that also employ shared split counters. The shared split counters could replace traditional 2-bit counters in other areas of computer microarchitecture design, for example in the classification table in data value prediction structures [18], or memory instruction dependence status prediction [19].

Branches tend to be biased toward being taken more often than not-taken (our simulations show that 60-70% of conditional branches are taken). This implies that the amount of information conveyed by the branch direction bit is also less than one bit per prediction. It may be possible to use coding techniques to further reduce the storage requirements of branch predictors without adversely affecting performance. Such an approach would probably require more complex hardware to perform the encoding and decoding.

Acknowledgments

This research was supported by NSF Grants CCR-9702281 and CCR-9985304.

References

[1] J. E. Smith, "A Study of Branch Prediction Strategies," in *Proceedings of the 8th International Symposium on Computer Architecture*, (Minneapolis, MN, USA), pp. 135–148, May 1981.

[2] D. A. Jiménez, S. W. Keckler, and C. Lin, "The Impact of Delay on the Design of Branch Predictors," in *Proceedings of the 33rd International Symposium on Microarchitecture*, (Monterey, CA, USA), pp. 4–13, December 2000.

[3] S. T. Pan, K. So, and J. T. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction using Branch Correlation," in *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems*, (Boston, MA, USA), pp. 12–15, October 1992.

[4] T.-Y. Yeh and Y. N. Patt, "Two-Level Adaptive Branch Prediction," in *Proceedings of the 24th International Symposium on Microarchitecture*, (Albuqueque, NM, USA), pp. 51–61, November 1991.

[5] T.-Y. Yeh and Y. N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction," in *Proceedings of the 19th International Symposium on Computer Architecture*, (Gold Coast, Australia), pp. 124–134, May 1992.

[6] T.-Y. Yeh and Y. N. Patt, "A Comparison of Dynamic Branch Predictors That Use Two Levels of Branch History," in *Proceedings of the 20th International Symposium on Computer Architecture*, pp. 257–266, 1993.

[7] S. McFarling, "Combining Branch Predictors," TN 36, Compaq Computer Corporation Western Research Laboratory, June 1993.

[8] A. N. Eden and T. N. Mudge, "The YAGS Branch Prediction Scheme," in *Proceedings of the 31st International Symposium on Microarchitecture*, (Dallas, TX, USA), pp. 69–77, November 1998.

[9] T. Juan, S. Sanjeevan, and J. J. Navarro, "Dynamic History-Length Fitting: A third level of adaptivity for branch prediction," in *Proceedings of the 25th International Symposium on Computer Architecture*, (Barcelona, Spain), pp. 156–166, June 1998.

[10] C.-C. Lee, I.-C. K. Chen, and T. N. Mudge, "The Bi-Mode Branch Predictor," in *Proceedings of the 30th International Symposium on Microarchitecture*, (Research Triangle Park, NC, USA), pp. 4–13, December 1997.

[11] S. Manne, A. Klauser, and D. Grunwald, "Branch Prediction using Selective Branch Inversion," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, (Newport Beach, CA, USA), pp. 81–110, October 1999.

[12] P. Michaud, A. Seznec, and R. Uhlig, "Trading Conflict and Capacity Aliasing in Conditional Branch Predictors," in *Proceedings of the 24th International Symposium on Computer Architecture*, (Boulder, CO, USA), pp. 292–303, June 1997.

[13] E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt, "The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference," in *Proceedings of the 24th International Symposium on Computer Architecture*, (Boulder, CO, USA), pp. 284–291, June 1997.

[14] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, "Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor," in *Proceedings of the 29th International Symposium on Computer Architecture*, (Anchorage, Alaska), May 2002.

[15] T. M. Austin, "SimpleScalar Hacker's Guide (for toolset release 2.0)," tech. rep., SimpleScalar LLC. `http://www.simplescalar.com/docs/hack_guide_v2.pdf`.

[16] D. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report 1342, University of Wisconsin, June 1997.

[17] K. Skadron, M. Martonosi, and D. W. Clark, "Selecting a Single, Representative Sample for Accurate Simulation of SPECint Benchmarks," TR 595-99, Princeton University Department of Computer Science, January 1999.

[18] M. H. Lipasti and J. P. Shen, "Exceeding the Dataflow Limit via Value Prediction," in *Proceedings of the 29th International Symposium on Microarchitecture*, (Paris, France), pp. 226–237, December 1996.

[19] A. Moshovos and G. S. Sohi, "Streamlining Inter-operation Memory Communication via Data Dependence Prediction," in *Proceedings of the 30th International Symposium on Microarchitecture*, (Research Triangle Park, NC, USA), pp. 235–245, December 1997.

Appendix A. Complete Misprediction Statistics

Table 7 in Section 5.4 presented the misprediction statistics for only a small number of benchmarks and the average case. In the three tables below, Table 8, Table 9 and Table 10, we present the full statistics for all of the benchmarks and all data sets simulated.

Benchmark (name.input)	Predictor	Mispredictions					Average Duel Len.	Avg. Trans. Duel Len.	Mispred. Rate
		Normal	Transient	Weak	Dueling	Trans. Duel			
bzip2.graphic	gshare	15600746	25794807	7401177	0	0	–	–	0.0488
	SSC2	14677214	25411914	7412410	445024	2077870	1.466	2.417	0.0500
	SSC4	13739802	25013427	7379926	1027842	3609406	1.523	1.871	0.0508
bzip2.program	gshare	27782616	29343845	12533910	0	0	–	–	0.0697
	SSC2	25874712	28530213	12986801	903637	3606307	1.336	2.389	0.0719
	SSC4	24154212	27770008	13362534	1849895	4842435	1.433	1.343	0.0720
bzip2.source	gshare	38717808	34610252	17142672	0	0	–	–	0.0905
	SSC2	36127186	33422880	17815476	1294047	5224576	1.313	2.486	0.0939
	SSC4	33771450	32331713	18303742	2624284	8709574	1.420	1.968	0.0957
crafty	gshare	39628812	27126473	11590284	0	0	–	–	0.0783
	SSC2	36762462	26678727	11107633	2354049	3923870	1.748	1.792	0.0808
	SSC4	33205346	26157534	10438849	5414993	10003314	2.117	2.175	0.0852
eon.cook	gshare	2968220	21827637	5574940	0	0	–	–	0.0304
	SSC2	2864426	23798318	1595182	837949	876107	170.280	18.450	0.0318
	SSC4	2778494	23727802	1634637	1834992	4248109	202.426	35.242	0.0379
eon.kajiya	gshare	27881052	31270346	13190067	0	0	–	–	0.0723
	SSC2	26800516	31059850	12234022	596966	3186054	1.527	2.324	0.0741
	SSC4	24501234	30695925	10762708	1487521	8598213	1.591	2.266	0.0764
eon.rushmeier	gshare	11881996	29783880	11881996	0	0	–	–	0.0478
	SSC2	11519428	29611481	6335743	862549	1221646	12.638	5.902	0.0500
	SSC4	10707646	29552001	5970518	2857201	5935364	12.891	8.543	0.0556
gap	gshare	18530318	27988385	7089962	0	0	–	–	0.0536
	SSC2	18007634	27893192	6916408	609408	2659308	4.857	7.510	0.0563
	SSC4	17328214	27866276	6142777	1854875	5919299	5.203	5.607	0.0594
gcc.166	gshare	10784448	14064655	4123134	0	0	–	–	0.0290
	SSC2	9952886	13927004	3817670	628143	1747995	2.445	2.509	0.0301
	SSC4	8846986	13714395	3484023	1389720	4408238	2.748	2.820	0.0319
gcc.200	gshare	26820794	25886992	9764982	0	0	–	–	0.0625
	SSC2	24973386	25613321	9267610	1636550	3439784	2.425	2.496	0.0649
	SSC4	22746016	25231121	8671110	3549125	8273471	2.812	2.725	0.0685

Table 8: Full misprediction classification statistics.

Benchmark (name.input)	Predictor	Mispredictions					Average Duel Len.	Avg. Trans. Duel Len.	Mispred. Rate
		Normal	Transient	Weak	Dueling	Trans. Duel			
gcc.expr	gshare	19949460	19825214	6773393	0	0	–	–	0.0465
	SSC2	18285754	19595165	6361120	1281721	2791153	2.184	2.276	0.0483
	SSC4	16114030	19245698	5851343	2876330	7011240	2.568	2.507	0.0511
gcc.integrate	gshare	15165070	17570938	5531430	0	0	–	–	0.0383
	SSC2	13951784	17358082	5160361	937540	2330592	2.139	2.322	0.0398
	SSC4	12277348	17016219	4783433	2096843	5694330	2.469	2.451	0.0419
gcc.scilab	gshare	27397434	26146337	9294856	0	0	–	–	0.0628
	SSC2	25276064	25791581	8902260	1906011	3841025	2.473	2.552	0.0657
	SSC4	22640922	25308836	8391917	4201666	9261735	2.874	2.738	0.0698
gzip.graphic	gshare	38521296	42577371	18469167	0	0	–	–	0.0996
	SSC2	37724088	42283593	18298502	607559	1603762	2.069	1.773	0.1005
	SSC4	36284924	42350426	16732947	1615590	4715546	2.107	1.851	0.1017
gzip.log	gshare	26101268	36037095	12653439	0	0	–	–	0.0748
	SSC2	25071650	35713936	12381794	579971	1706821	1.463	1.383	0.0755
	SSC4	23903780	35391939	11960188	1193068	3888889	1.606	1.524	0.0763
gzip.program	gshare	42418712	47131328	20365492	0	0	–	–	0.1099
	SSC2	40872458	46793582	19798748	892876	2463716	1.541	1.445	0.1108
	SSC4	39660388	46546106	19080131	1769387	4786168	1.647	1.496	0.1118
gzip.random	gshare	31151328	33955660	15555084	0	0	–	–	0.0807
	SSC2	31067716	33948868	15453692	83441	305790	2.021	2.908	0.0809
	SSC4	29499944	34160162	13480777	883483	3348454	1.747	1.689	0.0814
gzip.source	gshare	36057114	45219761	17269297	0	0	–	–	0.0985
	SSC2	34996454	44968894	16728265	659611	1891737	1.551	1.405	0.0992
	SSC4	33870678	44705418	16286990	1325090	3928630	1.639	1.531	0.1001
mcf	gshare	34774876	30148601	16797464	0	0	–	–	0.0844
	SSC2	34737464	30572864	17453720	884850	1774000	1.757	1.438	0.0854
	SSC4	33461412	30349553	16847331	2091893	4764199	2.093	2.029	0.0875
parser	gshare	29224810	22254628	9197872	0	0	–	–	0.0607
	SSC2	28221834	22132484	9187062	831446	1303697	2.053	2.447	0.0617
	SSC4	26829468	21841536	9203657	1950953	3842449	2.375	2.808	0.0637

Table 9: Full misprediction classification statistics (continued).

Benchmark (name.input)	Predictor	Mispredictions					Average Duel Len.	Avg. Trans. Duel Len.	Mispred. Rate
		Normal	Transient	Weak	Dueling	Trans. Duel			
perlbmk.makerand	gshare	158940	201827	82813	0	0	–	–	0.0025
	SSC2	159246	201709	82443	2638	3107	30.674	38.833	0.0039
	SSC4	159364	201720	81614	4514	6303	20.333	17.112	0.0193
perlbmk.splitmail	gshare	603458	1090732	324901	0	0	–	–	0.0244
	SSC2	569936	1078176	311621	113783	85101	8.500	2.091	0.0261
	SSC4	505550	1055548	331246	151183	233266	6.017	3.135	0.0277
perlbmk.scrabble	gshare	12281932	24062157	3849724	0	0	–	–	0.0402
	SSC2	11432104	23505033	4522104	267358	1814292	1.044	3.186	0.0444
	SSC4	10114254	23461194	3689625	1036983	7156744	1.362	4.853	0.0551
twolf	gshare	49054208	67809971	22903690	0	0	–	–	0.1398
	SSC2	45412470	66890150	21415010	1882889	6650131	1.388	1.583	0.1423
	SSC4	40601994	65703220	19903887	4159944	14670274	1.549	1.565	0.1451
vortex.1	gshare	8202454	14103166	3411132	0	0	–	–	0.0257
	SSC2	7981606	13971917	3252496	562426	2082799	5.738	6.843	0.0281
	SSC4	7417620	14291643	2187239	2401107	7010855	12.419	10.322	0.0343
vortex.2	gshare	7455970	15881228	2620189	0	0	–	–	0.0260
	SSC2	7214902	15802764	2386032	492700	1711968	4.869	5.698	0.0280
	SSC4	6658414	15572957	2466574	2091010	5838848	11.662	8.997	0.0347
vortex.3	gshare	7889126	13705032	3273231	0	0	–	–	0.0249
	SSC2	7677856	13665609	2953519	462633	1970255	5.071	6.823	0.0270
	SSC4	7155796	13667880	2544499	2161642	7476812	11.455	11.794	0.0341
vpr.place	gshare	17669474	16553885	7743471	0	0	–	–	0.1399
	SSC2	16628024	16309487	7510674	541950	1535060	1.384	1.472	0.1418
	SSC4	15698434	16029797	7243724	1122992	3211114	1.560	1.535	0.1444
vpr.route	gshare	18955822	31302341	7893428	0	0	–	–	0.0582
	SSC2	18227178	31404492	7254661	109119	1255252	1.395	2.143	0.0584
	SSC4	17772070	31386584	7014145	408181	2042800	2.745	2.137	0.0589
Average	gshare	22194123	26664639	9803559	0	0	–	–	0.0628
	SSC2	21140291	26480527	9272519	802374	2244268	–	–	0.0645
	SSC4	19738131	26218850	8766624	1980424	5635727	–	–	0.0680

Table 10: Full misprediction classification statistics (continued).