Alternative Implementations of Hybrid Branch Predictors

Po-Yung Chang  Eric Hao  Yale N. Patt
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

Abstract

Very accurate branch prediction is an important requirement for achieving high performance on deeply pipelined, superscalar processors. To improve on the prediction accuracy of current single-scheme branch predictors, hybrid (multiple-scheme) branch predictors have been proposed [6, 7]. These predictors combine multiple single-scheme predictors into a single predictor. They use a selection mechanism to decide for each branch, which single-scheme predictor to use. The performance of a hybrid predictor depends on its single-scheme predictor components and its selection mechanism. Using known single-scheme predictors and selection mechanisms, this paper identifies the most effective hybrid predictor implementation. In addition, it introduces a new selection mechanism, the 2-level selector, which further improves the performance of the hybrid branch predictor.

1 Introduction

Branches can significantly reduce the performance of high-performance processors. Speculative execution is one solution to the branch problem. However, all speculative work beyond a branch must be thrown away if that branch is mispredicted. Therefore, a very accurate branch prediction algorithm is important to high-performance microprocessors. As the number of transistors on a chip continues to increase (today, more than 9.3 million transistors on a single chip), more and more transistors can be allocated justifiably to satisfy this requirement.

Many dynamic prediction schemes have been studied [1, 2, 3, 4]. Simple history bit and counter-based prediction schemes [2] achieve prediction accuracies of 85%-90%. However, for today's wide-issue, deeply pipelined processors, a misprediction rate of 10% incurs a severe performance cost. For a four-wide processor with twelve stage pipelines, a 10% misprediction rate reduces performance by 55%, from a peak 4 instructions per cycle (IPC) to 1.8 (assuming one branch every four instructions). By keeping more history information, the misprediction rate can be halved to 5% [3, 4, 5, 6], improving performance to 2.5 IPC. However, the 5% misprediction rate still incurs a substantial performance penalty of 38%, wasting over one third of the processor's potential performance. Furthermore, as processors become wider and more deeply pipelined, this performance penalty will worsen.

To reduce this performance penalty, branch prediction accuracy must be further improved. Recently, hybrid branch predictors have been proposed as a way to achieve higher prediction accuracies [6, 7]. They combine multiple prediction schemes into a single predictor. A selection mechanism is used to decide for each branch, which single-scheme predictor to use. An effective hybrid branch predictor can exploit the different strengths of its single-scheme predictor components, enabling it to achieve a prediction accuracy greater than that which could be achieved by any of its components alone.

The performance of a hybrid branch predictor depends on the single-scheme predictors and the selection mechanism it uses. In this paper, we evaluate the performance of various implementations of the hybrid branch predictor. We consider different combinations of single-scheme predictors and different selection mechanisms. In addition, we propose a new selection mechanism, the 2-level branch predictor selector. By using more run-time information, it performs better than previously proposed selection mechanisms.

This paper is organized into five sections. Section 2 provides an overview of hybrid branch prediction. Section 3 compares the performance of various implementations. Section 4 introduces a new selection mechanism and compares its performance to that of previously proposed selection mechanisms. Section 5 provides some concluding remarks.

2 Hybrid Branch Predictors

Hybrid branch predictors were first proposed by McFarling [6]. They consist of a set of single-scheme predictors and a prediction selection mechanism. For each branch, each of the single-scheme predictors makes a prediction. The selection mechanism then chooses one of the predictions to be the hybrid predictor's prediction. McFarling proposed implementing the selection mechanism as an array of 2-bit counters. Each static branch was associated a counter which would keep track of which predictor was currently more accurate for that branch. We will refer to this array as the branch predictor selection table (BPST). Upon confirmation of a branch prediction, the counter would be incremented or decremented depending on which single-scheme predictor was correct. If both were correct (or incorrect), the counter state would be left unchanged.
Chang et al. [7] studied how hybrid predictors achieved their improved prediction accuracies. They showed that the optimal branch prediction scheme for each branch can be different because different branches have different dynamic behaviors. Thus, a higher prediction accuracy can be achieved if the branch predictor can use more than one prediction scheme and is able to select the most suitable scheme for each branch. Furthermore, they showed that certain branches can be predicted equally well by both simple and complex predictors, where complexity is a function of hardware cost. As a result, more cost effective predictors can be built if the majority of the hardware budget is dedicated to those branches that require complex predictors, leaving the minority share for those that require only simple predictors. Based on these results, they proposed a methodology, Branch Classification, for constructing hybrid branch predictors. Branch Classification partitions a program's branches into classes based on run-time and/or compile-time information. Each class is then associated with the most suitable prediction scheme from the hybrid predictor. Chang et al. proposed using dynamic taken rates as one possible means of classification. They showed that by dedicating most of the hardware budget for the complex predictors required by the branches that were not heavily biased in either the taken or not-taken direction, hybrid branch predictors were able to achieve higher prediction accuracies than those of single-scheme predictors.

3 Branch Predictor Configurations

An effective hybrid branch predictor combines the different strengths of its single-scheme predictor components to achieve a greater prediction accuracy. Combining two predictors that both achieve high prediction accuracies on the same subset of a program's branches will not yield a hybrid predictor with a significant increase in performance. In this section, the performance of various combinations of single-scheme predictors are compared in order to identify the most effective combinations.

Only hybrid branch predictor configurations consisting of two single-scheme predictors were considered. To identify the best configuration among them, every configuration in the design space must be considered. Because we could not see how to do this other than by simulating all configurations which would have been completely unmanageable, we opted for a process that assumed an idealized selection mechanism. This reduced the number of simulations that we had to consider to the number of single-scheme predictors. Our idealized selection mechanism operated as follows: Using foreknowledge of each single-scheme predictor's performance, we mapped each static branch to the single-scheme predictor (of the pair being considered) that would achieve the higher prediction accuracy for that branch over the entire run of the benchmark. With this assumption, the performance of a given configuration can be quickly calculated by examining the simulation results of its single-scheme predictor components (see section 3.2).

Note that while this selection mechanism achieves optimal performance for a static selector, it may not achieve optimal performance for a dynamic selector. A dynamic selector can choose a different predictor during different periods of a program's execution based on the particular dynamic information available at that time. This allows it to potentially achieve higher prediction accuracies for branches that under different circumstances are more accurately predicted by different predictors. However, the consistent ordering of predictor class combinations shown in our results (see section 3.3) gives support to the possibility that the best predictor class combination for the ideal static selector is also the best one for dynamic selectors.

3.1 Single-Scheme Predictors

For a given hardware cost, the hybrid predictor configuration specifies the classes of the single-scheme predictors used and the amount of hardware devoted to each scheme. The single-scheme predictors examined were divided into four classes:

1. static - the static branch predictor which bases its prediction on the direction the branch most frequently takes [8]. This is determined by profiling the program on a training input set.
2. 2bC(n) - the two bit counter predictor [2]. It consists of an array of n two bit counters. Each branch is mapped via its address to a counter which provides its prediction.
3. PAS(m,n) - a per-address variation of the Two-Level Adaptive Branch Predictor [4] consisting of 1K m-bit branch history registers and n pattern history tables.
4. gshare(m) - a modified version of the global variation of the Two-Level Adaptive Branch Predictor [4, 6] consisting of a single m-bit global branch history and a single pattern history table. The branch history and the branch address are XORed together to form the index into the pattern history table.

The static predictor was considered because it is a compile-time predictor which has no hardware cost. The 2bC predictor was considered because it is used by many of the current generation of commercial microprocessors. The PASs and gshare predictors were considered because they are variations of the highest-performing single-scheme predictor, the Two-Level Adaptive Branch Predictor [3, 11, 4, 5, 6].

For each predictor type, a range of predictor sizes were considered allowing us to vary the amount of hardware devoted to each scheme. The 2bC array size was varied from $2^{10}$ to $2^{20}$ entries. The branch history registers for the gshare and PASs schemes were varied from 10 to 20 bits. The number of pattern history tables for the PASs scheme was varied from 1 to 4. The hardware cost for each predictor type was estimated by the equations in table 1. Because different sizes were considered for each predictor class, it is possible for a given combination of predictor classes to have multiple representatives at a given level of hardware cost.
### Table 1: Hardware costs for the four classes of single-scheme predictors.

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Cost (Bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>static 2bC(n)</td>
<td>0</td>
</tr>
<tr>
<td>PAs(m,n)</td>
<td>2^n + m + 2^{m+1}n</td>
</tr>
<tr>
<td>gshare(m)</td>
<td>m + 2^{m+1}</td>
</tr>
</tbody>
</table>

### Table 2: Benchmarks simulated along with the input data sets used and the exact number of dynamic instructions simulated.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Input</th>
<th># of Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>espresso</td>
<td>bca.in</td>
<td>64944818</td>
</tr>
<tr>
<td>xlisp</td>
<td>li-input.lisp</td>
<td>134994603</td>
</tr>
<tr>
<td>eqnott</td>
<td>iml-prim3.eqn</td>
<td>105326183</td>
</tr>
<tr>
<td>compress</td>
<td>in</td>
<td>86445849</td>
</tr>
<tr>
<td>gcc</td>
<td>kmial</td>
<td>94334714</td>
</tr>
<tr>
<td></td>
<td>stmt.i</td>
<td>107162802</td>
</tr>
</tbody>
</table>

### 3.2 Simulation Methodology

The performance of each hybrid predictor configuration was measured by a combination of trace-driven simulation and post-simulation analysis. Each single-scheme predictor component of the hybrid predictor was first simulated by itself. The simulation recorded the number of correct and incorrect predictions for each static branch in the benchmark. To measure the performance of the hybrid predictor, the simulation results for the two single-scheme components were analyzed. For each static branch, the prediction results for the single-scheme component that achieved the higher prediction accuracy were used to represent the hybrid predictor’s performance for that branch, duplicating the behavior of the idealized selection mechanism.

The trace-driven simulator simulated the Motorola 88K ISA. The benchmarks simulated were the SPECint92 benchmarks. Each benchmark was simulated for 20 million conditional branches or until completion. Table 2 lists for each benchmark the reference data sets that was used for input and the exact number of instructions simulated.

### 3.3 Experimental Results

Every possible combination of single-scheme predictors from the set considered was simulated. Figure 1 lists for six levels of hardware cost (8KB–256KB) the misprediction rates of the best representative for each combination of single-scheme predictor classes. The best single-scheme predictor is included as well. The misprediction rates are the average of the rates achieved for the six SPECint92 benchmarks. The corresponding hardware cost level for each combination was determined by rounding its exact cost up to the next closest level.

At every level of hardware cost (with a minor exception at the 8KB level), the ordering of the predictor class combinations was the same with the gshare/PAs combination always achieving the lowest misprediction rate. This misprediction rate was on average 13% lower than that of its closest competitor, gshare/static. The best single scheme predictor at all levels of cost was gshare. Although it outperformed the PAs/static and 2bC hybrid combinations at cost levels above 8KB, it was outperformed by all the gshare hybrid combinations. The gshare/PAs combination was able to achieve the best performance because it was the only combination that effectively exploited both inter-branch and intra-branch correlation. The gshare component was able to accurately predict branches whose outcomes are dependent on the outcomes of other static branches. The PAs component was able to accurately predict branches whose outcomes are dependent on previous outcomes of the same static branch. Working together, they enabled the hybrid predictor to accurately predict a larger set of branches than what could be accurately predicted by a predictor that contained only one of the components.

When comparing the predictor combinations on a per-benchmark basis, the gshare/PAs combination was still always the best with the exception of the gcc benchmark. For gcc, the gshare/static and gshare/2bC achieved lower misprediction rates than gshare/PAs. Gcc’s results differed from the other benchmarks because it contains a large number of static branches in its working set. This large set can cause interference in the pattern history tables of the gshare and PAs predictors, reducing their ability to make accurate predictions [9, 10]. In addition, the large number of branches can incur a significant training cost. Both the gshare and PAs predictors must train themselves on the first few instances of the branch before they can begin to accurately predict it. Because both the static and 2bC schemes suffer little or no performance penalties due to interference or training, pairing either one with the gshare predictor produces a more effective hybrid predictor than gshare/PAs. Despite its weaknesses, gshare is still used as one of the components because there are still a significant number of branches in gcc for which gshare is the most effective predictor.
Table 3: Configurations for the best hybrid combination where *upwards* rounds cost to the next highest level and *nearest* rounds cost to the closest level.

<table>
<thead>
<tr>
<th>Cost (KB)</th>
<th>Rounding Model</th>
<th>Upwards</th>
<th>Nearest</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>gsh(14)/PAs(10,4)</td>
<td>gsh(15)/PAs(10,4)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>gsh(15)/PAs(12,4)</td>
<td>gsh(16)/PAs(12,4)</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>gsh(16)/PAs(13,4)</td>
<td>gsh(17)/PAs(13,4)</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>gsh(17)/PAs(16,4)</td>
<td>gsh(18)/PAs(16,4)</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>gsh(18)/PAs(15,4)</td>
<td>gsh(19)/PAs(15,4)</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>gsh(19)/PAs(16,4)</td>
<td>gsh(20)/PAs(16,4)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 lists the exact configurations for the best hybrid predictor (i.e., gshare/PAs combination) at each cost level. For every one of these combinations, half of the hardware was devoted to the gshare component and half was devoted to the PAs component. This result is due to the gshare component's cost and the hardware costs levels considered both always being a power of two. Thus, the configuration which maximizes the amount of hardware devoted to the gshare component while still leaving space for a PAs component is one that divides the hardware evenly. Table 3 also lists the exact configurations for the best hybrid predictors when each predictor's cost is rounded to the closest level instead of always upwards. In this case, the best combinations are the same as before except that the size of the gshare component is doubled so that the gshare component occupies two-thirds of the hardware budget. As in the upward rounding case, these configurations are the ones that maximize the gshare component size while still affording space for a PAs component.

4 Predictor Selection Mechanisms

The performance of a hybrid branch predictor also depends on its predictor selection mechanism. In the previous section, we have determined the optimal configurations of hybrid branch predictors with an idealized static selection mechanism. In this section, we will use these same configurations for evaluating the performance of our real selection mechanisms. We propose a new technique, the 2-level branch predictor selection mechanism, which uses more runtime information to improve the performance of the predictor selection mechanism.

4.1 2-level Branch Predictor Selection Algorithm

It is now well-known that the Two-Level Branch Predictor improves prediction accuracy over previously known single-level branch predictors [3]. The concepts embodied in the Two-Level Predictor can also be applied to the hybrid branch predictor selection mechanism. Figure 2 shows the structure of the 2-level predictor selection mechanism. A Branch History Register (BHR) holds the branch outcomes of the last $m$ branches encountered, where $m$ is the length of the BHR. This first level of branch history represents the state of branch execution when a branch is encountered. No extra hardware is required to maintain the first level of history if one of the component predictors already contains this information. That is, if the component predictor maintains a BHR, then the 2-level BPS mechanism does not need to maintain another copy of the BHR; instead, it just uses the component predictor's BHR. The Branch Predictor Selection Table (BPST) records which predictor was most frequently correct for the times this branch occurred with the associated branch history. This second level of history keeps track of the more accurate predictor for branches at different branch execution states.

When a branch is fetched, its instruction address and the current branch history is used to hash into the BPST. The associated counter is then used to select the appropriate prediction. By using the branch history to distinguish more execution states, 2-level predictor selection scheme can more accurately select the appropriate predictions.

Since the BPST and the PHT can be accessed in parallel, the time required for a hybrid predictor to make a prediction is $\max(BPST_{time}, PHT_{time}) + \text{mux}_{time}$ where $BPST_{time}$, $PHT_{time}$, and $\text{mux}_{time}$ are the access time of BPST, PHT, and mux respectively. Since the PHT are often much larger than the BPST, $PHT_{time}$ is greater than $BPST_{time}$. Thus, the time for making a prediction is $PHT_{time} + \text{mux}_{time}$, which is almost equivalent to the access time of single-scheme predictors.

Several variations of the 2-level predictor selection mechanism can be implemented. They are different in the manner in which the first level of branch history is kept and in the hashing function used to index into the BPST. For example, the first-level branch history can be Global, Per-set, or Per-address. In the Global history scheme, the first-level branch history is composed of the last $m$ branches encountered in the dynamic execution stream. In the Per-address scheme, the first-level branch history is composed of the last $m$ occurrences of the same branch instruction. In the Per-set scheme, the first-level branch history is composed of the last $m$ occurrences of branches in the same set. For the hashing function, we can, for example, XOR or concatenate the branch history with the instruction address.

4.2 Performance of the 2-level Branch Predictor Selection Mechanism

In this section, we will compare the performance of a 2-level history predictor selection mechanism with
that of a 2-bit counter history predictor selection mechanism.

Four variations of the 2-level predictor selection mechanism were studied. Two types of first-level branch history were examined: (1) Global and (2) Per-address. Two hashing functions were studied: (1) the branch history is XORed with the instruction address and (2) the branch history is concatenated with the instruction address. Table 4 summarizes these different configurations.

<table>
<thead>
<tr>
<th>Hash Function</th>
<th>History Type</th>
<th>Global</th>
<th>Per-address</th>
</tr>
</thead>
<tbody>
<tr>
<td>XOR</td>
<td>gXOR</td>
<td>pXOR</td>
<td></td>
</tr>
<tr>
<td>Concatenate</td>
<td>gCONC</td>
<td>pCONC</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Summary of 2-level predictor selection

For each one of these schemes, various branch history lengths can be used for indexing into the BPST. Rather than showing all these possibilities, figure 3 only shows the performance of the predictors with the highest accuracy across the benchmarks. The pCONC selection mechanism uses three bits of branch history, while all other methods use ten bits. The configuration of the hybrid branch predictor considered is gshare(16)/PAs(12,4), the best hybrid combination at implementation cost of 16Kbytes when predictors’ costs are rounds to the closest level. The BPST is a 1K-entry table of two bit counters. The misprediction rate in the graph indicates the rate at which the chosen predictor makes incorrect prediction.

With the exception of gcc, all four hashing schemes have similar performance. Young et al. [9] and Talcott et al. [10] have shown how branch prediction table interference can affect the performance of branch prediction schemes. Similarly, interference in the BPS table can affect the performance of the branch predictor selection mechanism. One advantage of using the branch history is that it reduces BPST interference. This can be accomplished by utilizing more of the BPST, reducing the number of branch instances that hash to a given BPST entry. For example, the espresso benchmark, with the 2-bit counter selection mechanism, utilizes 21.7% of the BPST. With 2-level selection mechanism such as gXOR, 92.2% of the BPST is utilized. This reduction in BPST interference results in gXOR’s higher prediction accuracy. For benchmarks where BPST interference is low, all four selection schemes have similar performance.

However, when there is already too much contention for the counters between different branches, as in gcc, the pCONC scheme outperforms the other three schemes. Conceptually, the pCONC scheme partitions the BPST into several sections. The branch address bits are used to select the appropriate section and the branch history bits are used to choose the appropriate counter within each section. The pCONC scheme with a small number branch history bits results in lower misprediction rates; a shorter BHR party.

Figure 3: Performance of Various 2-level BPS Mechanisms

Figure 4 compares the performance of the gXOR selection mechanism with that of the 2-bit counter branch predictor selection mechanism at various implementation costs. For illustration purposes, this figure also shows the performance of hybrid branch predictors with an ideal dynamic predictor selector and that of the best single-scheme predictor, gshare. The ideal dynamic predictor selection mechanism differs from the ideal static selection mechanism described in section 3. The ideal dynamic selector will at runtime always choose the component predictor that yields the correct prediction, if one exists, whereas the ideal static selector chooses the more suitable predictor for each branch at compile time. Although the gXOR mechanism outperforms the 2-bit counter predictor selection mechanism, it still operates far below its potential performance.

5 Conclusions

By combining multiple single-scheme branch predictors, hybrid branch predictors attempt to exploit the different strengths of different predictors. For this
attempt to result in significant increases in prediction accuracy, the hybrid predictor must combine an appropriate set of single-scheme predictors and use an effective predictor selection mechanism. This paper compared various hybrid predictor implementations to determine which single-scheme predictor combinations and branch selection mechanisms were most effective.

In determining the best single-scheme predictor combinations, this study assumed a static model of branch selection. Because of this assumption, this study did not fully exploit the capabilities of the hybrid predictor. The performance of the ideal dynamic selector (see figure 4) was significantly better than that of the ideal static selector (see figure 1). Thus, the predictor combinations used may not have been the optimal combinations for the dynamic selectors considered. Nonetheless, we believe that our experiments still yield valid information about the relative strengths of these selectors.

The hybrid predictor configurations considered in the study consisted of two single-scheme predictor components. Each component came from one of four classes: static, 2bC, PAs, and gshare. For an idealized static selection mechanism, the single-scheme predictor combination that achieved the lowest average mis-prediction rate for the SPECint92 benchmarks was the gshare/PAs combination. Its average mis-prediction rate was 13% lower than that of the next most effective combination. It was able to achieve this low mis-prediction rate because it was able to effectively exploit inter-branch correlation with the gshare component and intra-branch correlation with the PAs component. When pattern history table contention is a factor as in benchmarks with large numbers of static branches (e.g., gcc), the best combination was gshare/static, because the static component has no tables and does not suffer from this problem. For a fixed level of hardware cost, the gshare/PAs configurations that devoted the majority of the hardware to the gshare component achieved the lowest mis-prediction rates. Further work needs to be done to determine if these configurations are optimal for dynamic selection mechanisms as well.

To further improve the prediction accuracy of hybrid predictors, we introduced a new branch selection mechanism, the 2-level Branch Predictor Selector. It has four variations: gXOR, pXOR, gCONC, and pCONC. These variations use both branch history and the branch address to improve the accuracy of prediction. Our experiments showed that 2-level BPS outperforms 2-bit counter BPS and pCONC is more effective than pXOR, gCONC, and gXOR when significant BPST interference exists and the BPST is fully utilized. For the SPECint92 benchmarks, using the gXOR selection mechanism instead of the 2-bit counter BPS mechanism reduces the mis-prediction rate from 4.06% to 3.76%. For the gcc benchmark, where the BPST is partially utilized, the pCONC BPS mechanism consistently outperforms the 2-bit counter BPS mechanism because it reduces BPST interference, decreasing the mis-prediction rate by 5.45%.

Although the 2-level BPS mechanism introduced in this paper provides a performance increase over the 2-bit counter BPS mechanism, more effective BPS mechanisms are still required for the hybrid branch predictor to achieve its full potential performance. For a 16KB hybrid predictor, the 2-level BPS achieves a 4.1% mis-prediction rate whereas the ideal selector achieves a 2.1% mis-prediction rate, leaving significant room for improvement.

References