Branch Misprediction Prediction: Complementary Branch Predictors

Resit Sendag
Department of Electrical and Computer Engineering
University of Rhode Island
Kingston, Rhode Island

Joshua J. Yi
Networking and Computing Systems Group
Freescale Semiconductor, Inc.
Austin, The Great State of Texas

Peng-fei Chuang
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, Minnesota

Abstract[1] – **In this paper, we propose a new class of branch predictors, *complementary branch predictors*, which can be easily added to any branch predictor to improve the overall prediction accuracy. This mechanism differs from conventional branch predictors in that it focuses only on mispredicted branches. As a result, this mechanism has the advantages of scalability and flexibility (can be implemented with any branch predictor), but is not on the critical path. More specifically, this mechanism improves the branch prediction accuracy by predicting *which* future branch will be mispredicted next and *when* that will occur, and then it changes the predicted direction at the predicted time. Our results show that a branch predictor with the branch misprediction predictor achieves the same prediction accuracy as a conventional branch predictor that is 4 to 16 times larger, but without significantly increasing the overall complexity or lengthening the critical path.**

I. Introduction

Typically, increased branch prediction accuracy comes at the cost of increased complexity (*e.g.*, more complex algorithms) and chip area (*e.g.*, larger tables). Unfortunately, this may result in higher prediction latencies and an increased misprediction penalty which may ultimately offset the higher prediction accuracy, resulting in a net performance loss [2]. By contrast, in this paper, we propose a complementary branch prediction mechanism that is designed to be both scalable and flexible, does not affect the prediction latency, and is not the branch predictor's critical path. The reason that our mechanism has these characteristics is because it targets mispredicted branches only, which are the subset of branches that the branch predictor does not predict accurately. Our mechanism complements any branch predictor in that it strengthens the weakness of the branch predictor, namely, frequently mispredicted branches which are either due to the limitations of the branch predictor's algorithm or its implementation.

More specifically, this mechanism uses the *branch misprediction history* to predict **which** future branch will mispredict next and **when** that will occur. Then, before the misprediction actually occurs, the branch misprediction predictor (BMP) changes the prediction to avoid a misprediction and the subsequent recovery. Since it only focuses on frequently mispredicted branches, it has the advantage of flexibility in that it can improve the branch prediction accuracy of any branch predictor, static or dynamic, simple or complex. And since it targets future branches only, the BMP can be added to any processor without significantly changing the current branch predictor or the pipeline. Finally, in contrast with the state-of-the-art, the BMP is not on the processor's critical path because it: 1) Is accessed only after a branch misprediction and 2) Makes predictions for future branches only. Consequently, it does not affect the prediction latency, which allows for scalability.

To demonstrate the efficacy of this new class of branch predictors, we implemented a simple, but fully functional, BMP. Based on our results, we make the following conclusions about complementary branch predictors such as the BMP:

1. It yields A) higher prediction accuracy without significantly increasing the size or latency of the predictor or B) the same accuracy with a lower area and power cost.
2. It significantly improves the branch prediction accuracy of all branch predictors that we tested, from relatively simple gshare predictors to complex neural predictors, for the SPEC CPU2000 benchmarks.
3. It is more scalable since it is not on the critical path of the branch predictor and will not increase the number of front-end pipeline stages.

II. Predicting Branch Mispredictions

Fundamentally, the BMP operates by determining patterns of branch mispredictions and correcting future, incorrect predictions. The remainder of this section describes how the BMP detects and corrects future mispredictions, a simple implementation, and how it interacts with a conventional branch predictor.

A. *Description, Operation, and Implementation of the Mispredicted Branch Table*

The key component of the BMP is the mispredicted branch table (MPBT). We form the index to the MPBT by XOR-ing the PC of current mispredicted branch with global history bits and with a concatenation of the misprediction history bits and the branch misprediction distance (the number of committed branches between the last two branch mispredictions).

The output of the MPBT is a prediction of the distance to and address of the next-to-be-mispredicted branch. Note that this output is fundamentally different than the predicted direction that is the output of conventional branch predictors.

After every misprediction, the BMP forms the index and accesses the MPBT to predict when – in terms of the number of branches – the branch predictor will mispredict

[1] This work was supported in part by National Science Foundation Grant CCF-0541162.
Manuscript submitted: 01-Jul-2007. Manuscript accepted: 07-Aug-2007. Final manuscript received: 12-Sep-2007.

again. The BMP decrements the predicted distance counter every time a branch instruction is fetched. When the predicted distance is zero, the predicted branch address is compared against the PC of the current branch; if they match, the BMP corrects the branch prediction.

***MPBT implementation and prediction*:** The width of each MPBT entry is 14 bits wide; 4 bits for the address, 8 bits for the distance, 1 used bit, and 1 T/NT bit. These values work sufficiently well for the SPEC CPU2000 benchmarks. After indexing the MPBT, the entry corresponding to the index is copied into two registers, one of which stores the 4-bit next-to-be-mispredicted PC (NMPC) and the 1-bit T/NT field, while the other stores the 8-bit misprediction distance (MPD). The NMPC register and T/NT field update only after a branch misprediction. By contrast, the MPD register value decrements after each fetched branch instruction until the next misprediction occurs (at which point the counter is set to the next predicted misprediction distance) or it decrements to zero, whichever happens first. In the former case, the BMP has overestimated the distance, so its prediction of a branch misprediction is ignored. Note that this does not result in any additional branch mispredictions.

***Updating the MPBT*:** After a branch misprediction, the 8-bit branch counter (BC) starts counting the number of committed branches between consecutive mispredictions. After the next misprediction, the BMP updates the MPBT entry that made the last prediction with the value of the BC and bits 3 to 6 of the PC of the mispredicted branch, and the T/NT field is updated with the predicted direction (which was wrong).

Correct BMP predictions set the used bit for that entry. If the used bit is set, but the MPBT prediction is wrong, the entry is not updated and the used bit is reset. This approach protects MPBT entries from eviction based on a single misprediction. However, this entry will be evicted if it causes two consecutive incorrect predictions of branch mispredictions. The used bit also alleviates any aliasing to the same entry.

B. *Interaction with the branch predictor*

The BMP only interacts with the branch predictor for a branch that is predicted to be mispredicted, which is significantly less often than a confidence estimator does (*i.e.*, on every predicted branch). Since the BMP's prediction is made well ahead of the time, different optimizations can be used to eliminate the extra latency due to correcting a future prediction. One possible optimization would be to time any actions to finish at the same time that the MPD value reaches zero. However, specific optimizations of the basic BMP design are out of the scope for this paper.

III. Results and Analysis

To evaluate the efficacy of our BMP mechanism, we added it to the gshare [5], PAp [7], Alpha 21264 [4] ("Alpha"), and piecewise linear [3] ("PWL") branch predictors and evaluated it for all 26 SPEC CPU2000 benchmarks.

A. *Branch Misprediction Predictor Performance*

Figures 1 and 2 show the reduction in the branch misprediction rate due to using the 64, 256, 1024, and 4096 entry BMPs, for the integer and floating-point benchmarks, respectively. The baseline size of the branch predictors is 8KB.

The results show that using the BMP mechanism significantly reduces the number of branch mispredictions for all branch predictors and for all benchmarks. More specifically, adding a 64-entry BMP reduces the branch misprediction rate by 13.9% (*alpha*) to 20.0% (*gshare*) for the SPECint benchmarks and 53.0% (*PWL*) to 66.9% (*gshare*) for the SPECfp benchmarks. For *swim*, *mgrid, applu, art,* and *lucas*, the BMP is able to eliminate over 90% of the branch mispredictions for at least one branch predictor. Based on these results, we conclude that our BMP mechanism is able to significantly reduce the branch misprediction rate of both simple and complex branch predictors, which, in addition to its relatively small size and low complexity, makes it a very appealing complement to virtually any branch predictor.

To compute the BMP prediction accuracy, we divide the number of successfully corrected predictions by the sum of the number of successfully and unsuccessfully corrected predictions. For most BMP sizes and branch predictors, the prediction accuracy is over 90% which means that the BMP is extremely accurate in predicting both which future branch will mispredict next and when it will be mispredicted.

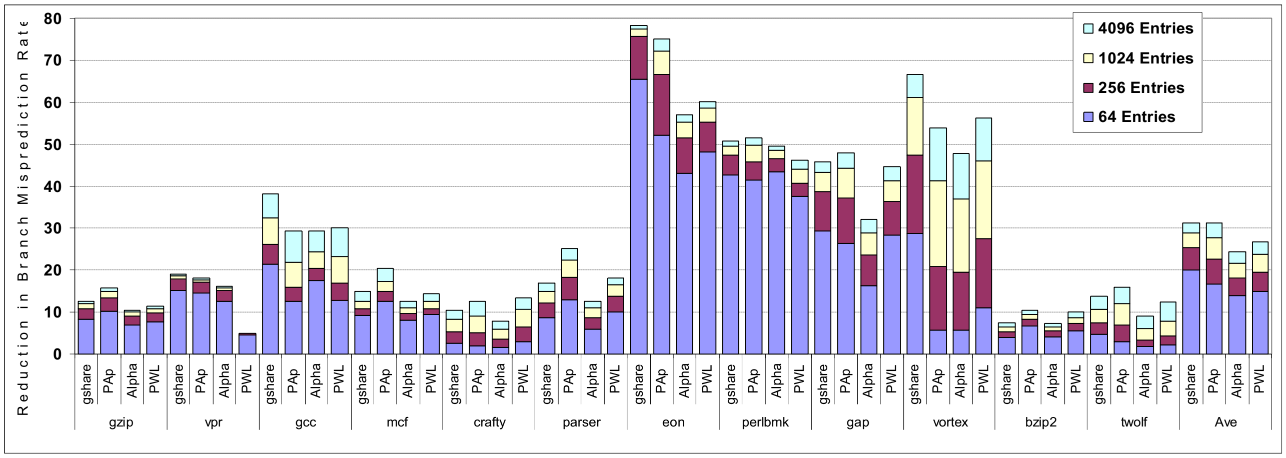


Figure 1. Percentage reduction in the branch misprediction rate due to BMP for SPECint

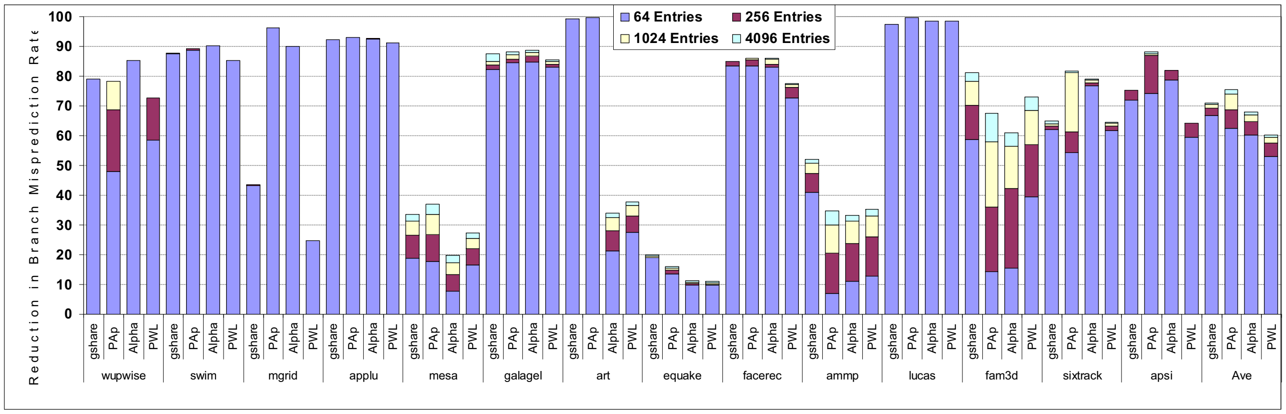


Figure 2. Percentage reduction in the branch misprediction rate due to BMP for SPECfp

B. *Hardware Budget vs. Branch Prediction Accuracy: Comparing the BMP vs. Larger Branch Predictors*

Table 1 shows the hardware budget that each branch predictor needs to have to match or exceed the branch prediction accuracy for branch predictors that have the BMP mechanism. The total hardware budget of the BP+BMP predictor is less than 4KB (BP+BMP: 2KB+1.8KB) The results in Table 1 show that for all benchmark suites, a branch predictor without a BMP mechanism needs to have a hardware budget that is at least four times as large as the branch predictor with the BMP mechanism. For the SPECfp benchmarks, matching the branch prediction accuracy of the BMP-based branch predictor of 4KB requires a branch predictor with a hardware budget of more than 64KB. However, using such a large branch predictor would most likely significantly increase the prediction latency, thus offsetting the performance gains due to the higher branch prediction accuracy. By contrast, adding the BMP mechanism to a smaller, faster branch predictor can achieve the same or higher prediction accuracies without incurring higher prediction latencies and power consumption.

Table 1. Hardware budget of branch predictor needed to match/exceed the branch prediction accuracy of a 2KB branch predictor with a 1024-entry BMP (Total budget of < 4KB)

Suite	gshare	pap	Alpha	PWL
Int	16KB	> 64KB	16KB	32KB
FP	> 64KB	> 64KB	> 64KB	> 64KB

C. *Comparing the BMP vs. Loop Predictors*

At first glance, the BMP may appear to be another type of loop predictor (LP) [1]. However, there are several significant differences between the BMP and a LP. First, the BMP is not limited only to loops, but rather can target all types of branches. Second, the LP only uses local history to make predictions while the BMP uses different types of global history (*e.g.*, global misprediction history) to make its predictions. Finally, since the LP competes with other constituent predictors within the branch predictor, even if it is chosen as the highest confidence prediction, its prediction may not be different than the predictions from the other constituent predictors, *i.e.*, all predictors may make a correct prediction. By contrast, the BMP only makes predictions for frequently mispredicted branches. To quantify the performance difference between a BMP and LP, we implemented the loop predictor that was described in [6].

Figure 3 shows the breakdown of branches, as percentage of all branch mispredictions (i.e., when the LP and BMP are not used), that were successfully predicted by the LP, but not by the BMP (bottom-most segment – "LP Only"); the BMP, but not by the LP (top-most segment – "BMP Only"); and by both the BMP and LP (middle segment – "BMP and LP Both Correct"). Therefore, the height of the bottom two segments shows the percentage of mispredictions that were avoided due to using the LP, while the height of the top two segments shows the percentage of mispredictions that were successfully corrected by the BMP. The size of both the LP and BMP was 4KB while the baseline branch predictor size was 8KB.

For the integer benchmarks, the results in Figure 3 show that the LP corrects a very small number of mispredictions (less than 2.3% for all branch predictors) that the BMP does not correctly predict. Overall, the bottom two segments show that the LP corrects fewer than 8.1% of the mispredictions. By contrast, the BMP corrects at least 23% of all mispredictions, including 18% of all mispredictions that the LP does not correctly predict.

The results for the floating-point benchmarks are similar, with exception that the LP corrects a much higher number of mispredictions. Nevertheless, the BMP outperforms the LP for all predictors.

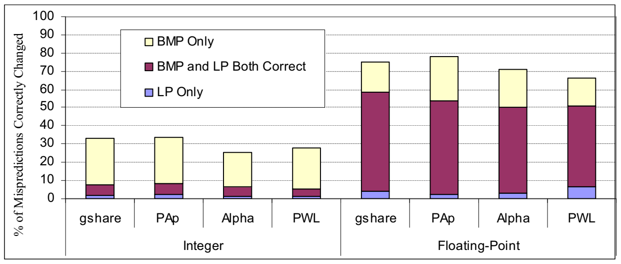
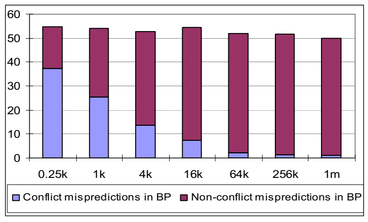


Figure 3. Breakdown of branches that were successfully predicted due to using a LP or a BMP as a percentage of the total number of mispredictions. Both LP and BMP are 4KB. Baseline BP is 8KB.

On average, 21% and 65% of the mispredictions eliminated by BMP are due to loops, which a loop predictor can also correct, for SPEC integer and floating-point benchmarks, respectively.

IV. Investigating Why BMP Works

To investigate why and where a BMP helps to correct branch mispredictions, we first check whether the BMP primarily corrects mispredictions that are the result of conflicts in branch predictor table, *i.e.*, aliasing. Conflicts occur when multiple branch-history pairs share the same location in the branch predictor table. Figure 4a shows percentage of branch mispredictions corrected by BMP that are due to the conflicts for varying sizes of *gshare* branch predictor. This figure shows the average behavior of 8 selected SPEC benchmarks (*gcc, eon, perl, gap, vortex, mesa, fma3d,* and *apsi*) where BMP does very well. To filter out the impact of fixed loop counts, benchmarks where both BMP and a loop predictor do well are not included. Figure 4a shows that, for a constant 0.5KB size BMP, the percentage of mispredictions corrected by BMP that are due to conflicts in the branch predictor table decreases as the branch predictor size increases, from 60% when using 0.25KB BP to less than 4% for 64KB and larger BPs. Since the percentage of the mispredictions due to conflicts decreases dramatically for increasing branch predictor sizes (From 68% for 0.25KB *gshare* to 1.8% for 1MB *gshare* (not shown)) and since the BMP can reduce the overall branch misprediction rate by about 50% for different sizes of the *gshare* predictor, as shown in Figure 4a, we conclude that the BMP does not primarily correct mispredictions that are due to conflicts in the branch predictor tables only, but does correct mispredictions due to other non-capacity-based reasons.



```
....
while (true) {
    N = 150;
    for (i = 0; i < N; i++)
      { /* This is a loop with
        alternating loop counts */ }
    N = 250    N;
}
....
```

(a) (b)

Figure 4. (a) Percentage reduction in misprediction rate with a 0.5KB BMP for varying sizes of *gshare* BP. (b) A code example where BMP works well.

To understand the causes of branch mispredictions that are not due to conflicts, we analyze profile data and the source code for the 8 SPEC 2000 benchmarks. We observed that in these benchmarks, 30% to 60% of the mispredictions that are corrected by BMP are due to loop branches that have *varying* loop counts, which are longer than what a branch predictor can distinguish, or have early loop exits, such as a `break` in a `for` or a `while` loop. Figure 4b presents a synthetic, but representative, code snippet. Variations of this example code occur in all benchmarks (see [8]), often with unstable loop counts or early loop exits. This example shows a simple loop whose loop count alternates between 100 and 150. The exit branch for the `for` loop will be mispredicted for as many times as the `while` loop condition is true. In the example, 150 bits of history is needed to eliminate the mispredictions at the loop exit. Simulations confirm that various branch predictors with a 256KB hardware budget and an 8KB loop predictor mispredicted the loop exit branch every time. By contrast, a 4-entry BMP can easily correct all of these mispredictions by predicting the next misprediction distance, which is either 101 or 151.

Another example where the BMP works is an early exit branch inside a `for` loop, which further complicates the branch history. This type of behavior is also often seen in the benchmarks that we studied. The early exit branch inside the `for` loop will be mispredicted often when it is taken. In the same manner that the BMP corrected mispredictions for the `for` loop in Figure 4b, BMP can also correct this type of misprediction, while the other branch predictors that we tested do not.

In summary, the code in Figure 4b shows that a BMP is an alternative approach to exploit long branch histories. While some advanced branch predictors have been proposed, such as neural predictors [3] or O-GEHL predictor [9], they are much more complex and larger than a simple BMP. BMP complements branch predictors to exploit very long histories without significant hardware complexity and delay.

The ability to provide longer history and to help alleviate branch predictor table conflicts is not the only way in which a BMP correct mispredictions. A detailed analysis of the misprediction patterns and why branch predictors fail to exploit these patterns need further research, and left as future work.

V. Conclusion and Future Work

In this paper, we present the branch misprediction predictor (BMP). BMP tracks the misprediction histories of branches at the local and global levels and uses this information to predict when the next branch misprediction will occur and for which branch. Then, at the predicted time, the BMP changes the prediction for the specific branch. By focusing on future branch mispredictions, BMP is off the processor's critical path, thereby simultaneously increases the branch prediction accuracy without increasing the prediction latency. Our results show that the BMP can significantly reduce the branch misprediction rate for both simple and complex branch predictors. To achieve the same prediction accuracy, branch predictors without a BMP require hardware budgets that are 4 to 16 times larger.

The BMP described in this paper was just a proof-of-concept to validate the efficacy of complementary branch predictors. However, a detailed analysis of the misprediction patterns and why branch predictors fail to exploit these patterns need further research. In the future, we plan to: 1) further investigate the causes of misprediction patterns 2) explore alternative BMP indexes, and 3) evaluate the effects on IPC performance and power.

VI. References

[1] Gochman *et al.*, "The Intel PentiumM processor: Microarchitecture and performance," *Intel Tech. Journal*, 7(2), pp. 21-33, 2003.
[2] D. Jiménez, "Reconsidering complex branch predictors," *HPCA 2003*.
[3] D. Jiménez, "Piecewise Linear branch prediction," *ISCA 2005*.
[4] R. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, 1999.
[5] S. McFarling, "Combining Branch Predictors," Digital Res. Lab. Tech. Rep. TN-36M, 1993.
[6] A. Seznec, "A 256 Kbits L-TAGE branch predictor," *JILP,* 2006.
[7] T. Yeh and Y. Patt, "Two-Level Adaptive Branch Prediction," *MICRO 1991*.
[8] Sendag *et al.*, "Branch Misprediction Patterns," *URI Tech. Rep.*, 2007.
[9] A. Seznec, "Analysis of the O-GEHL predictor," *ISCA 2005.*