Correlation-Based Branch Prediction

Shien-Tai Pan  
Advanced Workstation Division  
IBM Corporation, 9440  
11400 Burnet Road  
Austin, Texas 78758  
stpan@hwperform.austin.ibm.com

Kimming So  
Advanced Workstation Division  
IBM Corporation, 4305  
11400 Burnet Road  
Austin, Texas 78758  
sck@watson.ibm.com

Joseph T. Rahmeh  
Electrical and Computer Engineering Department  
University of Texas at Austin  
Austin, Texas 78712  
rahmeh@cerc.utexas.edu

Abstract

Most dynamic branch prediction schemes reported in the literature predict the outcome of a branch based exclusively on the past behavior of the branch. We have found that in many programs where control-flows are intensive, very often the outcome of a branch is also affected by previously executed branches, i.e., by "correlated branches". To enhance the accuracy of branch prediction, we propose a "correlation-based" branch prediction scheme which takes into consideration the information provided by the outcomes of other branches as well. The proposed scheme is examined using traces collected from running the SPEC benchmark suite on an IBM RISC System/6000™ workstation. The results show that up to an addition of 11% prediction accuracy is achievable by the new scheme. To alleviate the branch problem using a Branch Target Buffer (BTB), further reduction in branch delay is attained when the new scheme is implemented in conjunction with the BTB.

1. Introduction

Microprocessor technology has made tremendous progress in the past decade. With the advent of RISC architecture and the advances in VLSI technology, computer designers are able to exploit more instruction-level parallelism by using deeper pipelines and more execution units [1, 2]. As sophisticated hardware is built to exploit the available instruction-level parallelism, more attention needs to be paid to the pipeline delay as a result of branch instruction execution [9].

Dynamic branch prediction, reported elsewhere [3–8], has been shown to be an effective way of reducing the branch delay. With this approach, a table, called the branch prediction table (BPT), is required to store the necessary information for predicting the likely outcomes of branches. The table is accessed using a hashed index from the branch address under consideration. The prediction information thus obtained usually serves as an input to a finite-state machine (FSM) which performs the prediction for the branch. The N-bit counter branch prediction scheme [3, 5, 8], in particular, uses an N-bit up/down counter to represent the state of the FSM.

The counter is incremented or decremented according to whether or not the branch is taken. We refer to this scheme as the counter-based branch prediction. Reducing the branch delay using a branch target buffer (BTB) is also popular [3–6]. A BTB, typically organized as a set-associative cache, reduces the branch delay by storing in the buffer the branch target addresses and the instructions at each branch target address. There are many variations in the design of a BTB. Some of the designs also require to predict the outcome of a branch based on its dynamic behavior.

A common limitation with most of the dynamic branch prediction schemes is that the prediction is based exclusively on the past behavior of the branch under consideration, completely ignoring the information provided by the executions of other branches. We have found that in many programs where control-flows are intensive, very often the outcome of a branch is also affected by previously executed branches, i.e., by correlated branches. In this paper, we propose a correlation-based branch prediction scheme and show that how the prediction accuracy can be improved by making the use of branch correlation.

The proposed new scheme is evaluated using traces collected from running the SPEC benchmark suite [10] on an IBM RISC System/6000 workstation. The results show that up to an addition of 11% prediction accuracy is achievable by the new scheme. To alleviate the branch problem using a BTB, further reduction in branch delay is attained when the new scheme is implemented in conjunction with the BTB.

The remainder of this paper is organized as follows: In section 2, we give a brief overview of the counter-based branch prediction. In section 3, we introduce the correlation-based branch prediction scheme. Its implementation is given in section 4. In section 5, simulation results are presented. In section 6, we give the conclusions.

2. Overview of Counter-Based Branch Prediction

The basic idea of the counter-based branch prediction is the use of N-bit up/down counters to record the prediction information [8]. In the ideal case, an N-bit counter (with some initial value) is assigned to each branch. When a branch is
about to be executed, the counter value C, associated with that branch, is used for prediction. If C is greater than or equal to a predetermined threshold value, L, the branch is predicted taken, otherwise it is predicted not taken. A typical value for L is $2^N - 1$. The counter value C is updated whenever that branch is resolved. If the branch is taken, C is incremented by one, otherwise it is decremented by one. If C is zero, it remains at zero as long as the branch is not taken. The operation of the N-bit counter scheme corresponds to a path, meaning that the branch "taken path" is the path for which the condition is true. Since the outcome of b3 depends on the values of aa and bb, it is obvious that b3 is correlated with both b1 and b2.

```
if (aa==2) /* b1 */
   aa = 0;
if (bb==2) /* b2 */
   bb = 0;
if (aa != bb) /* b3 */
   ....
```

Fig. 2 A Code Fragment from SPEC: eqntott

Fig. 3 Branch Tree for the Code Given in Fig. 2

Fi g. 3 shows the part of the branch tree before the execution of b3 given that b1 and b2 have been executed. There are four possible paths reaching b3 through the executions of b1 and b2. For example, if b1 is taken and b2 is not taken, then b3 is reached via the 1–0 path. It is clear that if b3 is reached via the 0–0 path, the outcome of b3 is deterministic.

The example suggests that the outcome of a branch can be more readily determined once the path leading to it is known. By splitting the branch history of b3 into four subhistories according to the branch paths leading to it, one may reduce the randomness of the history of b3 and make a better prediction. This observation leads to the idea of the correlation-based branch prediction. Basically, the idea is to use the branch path information to split the global history of a branch into several subhistories and selectively choose a proper subhistory for predicting the outcome of the branch. When a 2-bit counter is used for prediction within each subhistory of b3, four FSM's are created, with one for each subhistory. This is shown in Fig. 4. Note that a 2–bit shift register is required to store the outcomes of the most recent two branches. The value of the 2–bit shift register represents one of the four branch paths leading to b3. This register will be used to select the appropriate FSM for predicting the outcome of b3.

```
path 0-0
path 0-1
path 1-0
path 1-1
```

Fig. 4 FSM's using Four 2-bit Counters

Generally, an M-step correlation-based branch prediction uses an M-bit shift register to store the outcomes of the most recent M branches (including unconditional branches). This M-bit shift register is able to identify a total of $2^M$ subhistories of a branch. The prediction is done independently within each subhistory using any (or the best) history-based branch prediction. A good candidate for this is the N-bit counter-based branch prediction. In this case, a total of $2^M$ FSM's are created, with each one independently implementing the N-bit counter branch prediction. Everytime the outcome of a branch is to be predicted, the M–bit shift register selects a proper FSM for prediction. The state transition and the state update for the selected FSM are done according to the N–bit counter–based prediction algorithm. In the following, we refer to this scheme as the (M,N) correlation–based branch prediction scheme or simply the (M,N) correlation scheme, meaning that an M–bit shift register is used to select an N–bit counter for prediction.

4. Implementation

When implementing the N-bit counter or the (M,N) correlation scheme, a BPT is required to store the prediction information. Fig. 5 (a) shows the logical organization of a 1K–entry BPT for the 2–bit counter scheme, with each entry
containing 2 prediction bits. Fig. 5 (b) shows the logical organization of a 1K-entry BPT for the (2,2) correlation scheme, which has entry containing 2x2^2=8 prediction bits. Notice the difference in physical size of the two tables. General, if a 2^L-entry table is used for the (M,N) correlation scheme, a total of N x 2^M bits is required for the table, with each entry containing 2^M sets of N prediction bits. Usually, the table is accessed using some low-order I bits of the branch address. Once the entry is determined, the M-bit shift register which stores the outcomes of the most recent M branches is used to select the proper set of N bits from the entry. These N bits are used to predict the outcome of the branch.

(a) 2-bit counter scheme   (b) (2,2) correlation scheme

**Fig. 5 Logical Organization of a 1K-Entry Table**

While the logical organization of the tables for the counter and the correlation schemes are different, the physical implementations are quite similar. Fig. 6 shows the implementation using a 1KB-BPT. When this table is used for the 2-bit counter scheme, 12 bits are required from the branch address for table lookup (Fig. 6 (a)). However, if the same table is used for correlation schemes, some of the bits for table lookup are obtained from the shift-register. For example, if the (8,2) correlation scheme is implemented using the same table, the index for table lookup consist of 8 bits from the shift register and 4 bits from the branch address.

(a) 2-bit counter scheme   (b) (8,2) correlation scheme

**Fig. 6 Physical Implementation for a 1KB-BPT**

The correlation-based branch prediction can also be implemented in conjunction with a BTB. Fig. 7 shows one possible implementation. Its operation is explain below. Note that Fig. 7 is used only to explain the idea. The actual implementation is not limited to the one shown in the figure.

During the instruction fetch cycle, both tables are accessed using the instruction fetch address (IFA). The branch addresses (b0b1's) stored in the BTB are compared (shown as "cpr" in Fig. 7) in parallel with the IFA. If a match is found (hit), the corresponding target address (t@) stored in the BTB is used to redirect the next instruction fetch based on the predicted result from the BPT (determined by "b0, b1"). If there is no match (miss) or "not taken" is predicted by the BPT, the next instruction fetch continues along the sequential path. Note that both tables are indepedently updated as usual.

**Fig. 7 A 4-way BTB Implemented with a BPT**

One of the advantages of separating the prediction from the BTB is that a moderately large BPT can greatly improve the prediction accuracy without increasing the size of the BTB. In real implementation, some reductions in branch delay is possible even when a correctly predicted taken branch has a wrong target address stored in the BTB.

5. Simulation Results

Trace-driven simulations are used to evaluate the proposed correlation scheme. Seven of the SPEC benchmarks are used for the evaluation: **doduc, spice, fpppp, gcc, espresso, eqntott, and li**. The traces are collected using a trace program and commercially available C and FORTRAN compilers for the IBM RISC System/6000 system. Each trace contains fifty million executed instructions. The accuracy used for measuring the efficiency of the branch prediction is defined as the percentage of correct predictions, when a BPT is used alone, or the percentage of correct predictions resulting in correct redirections of the next instruction fetch, when a BPT is used in conjunction with a BTB. Note that a BTB can be implemented by itself. When this is the case, we simulate the BTB which only keeps the information of taken branches and uses the default branch prediction rules (i.e., predict taken when an address hits the BTB; not taken, otherwise).

5.1 Accuracy for a BPT implemented alone

Fig. 8 shows the results for a BPT with size fixed at 1KB. The figure compares the accuracy obtained by implementing the 2-bit counter scheme and the additional accuracy gained by implementing the (8, 2) correlation scheme. Since the 2-bit counter scheme has already provided very high accuracies for **doduc** and **espresso** (about 95%), there is very little
chance for the correlation scheme to gain more accuracy. The benchmark gcc shows very little improvement in accuracy. This is because that a 1KB-table is not large enough to hold most of the frequently executed branches in gcc.

Although we have only shown the results for the 8-step correlation scheme, it is observed from the simulation that, as the number of table entries is fixed, the accuracy increases as the number of correlation steps increases. This observation is true for all the seven benchmarks.

When the table is large enough to contain most of the branches, the prediction accuracy of the 2-bit counter scheme starts saturating. The accuracy asymptotically approaches the limit shown in Fig. 9. As we mentioned earlier, one of the limitations of the counter-based scheme is that it is self-history based. Since the correlation scheme provides better prediction by incorporating the correlated information from other branches, higher accuracy is attainable. Fig. 9 also shows the maximum achievable accuracy by the correlation scheme.

5.2 Accuracy for a BTB implemented with a BPT

Fig. 10 shows the results for a 512-entry direct-mapped BTB. The figure compares the accuracies for the BTB implemented with and without a BPT. When the BTB is implemented with a BPT, both 2-bit counter and correlation predictions are compared. Note that the accuracy in this case is defined as the percentage of correct predictions that result in correct next instruction fetches.

The remaining benchmarks show considerable improvements in accuracy. The two biggest gains in accuracy are obtained by eqntot and li. Since branches in eqntot are highly correlated, the 2-bit counter scheme cannot provide high accuracy (only about 83%). However, an additional 11% accuracy can be attained by the correlation scheme. The second highest improvement in accuracy is achieved by li (more than 5%). It is known that li is a “pointer-chasing” oriented program where a typical compiler may generate load, compare, and branch instructions in sequence over and over again. The branch correlation exists wherever the data loaded for determining the branch direction is affected by the directions of prior branches.

Although we have only shown the results for the 8-step correlation scheme, it is observed from the simulation that, as the number of table entries is fixed, the accuracy increases as the number of correlation steps increases. This observation is true for all the seven benchmarks.

When the table is large enough to contain most of the branches, the prediction accuracy of the 2-bit counter scheme starts saturating. The accuracy asymptotically approaches the limit shown in Fig. 9. As we mentioned earlier, one of the limitations of the counter-based scheme is that it is self-history based. Since the correlation scheme provides better prediction by incorporating the correlated information from other branches, higher accuracy is attainable. Fig. 9 also shows the maximum achievable accuracy by the correlation scheme.
employed in the BPT (2-bit counter provides only 83% correct predictions for eqnot). However, since the correlation scheme is able to provide more correct predictions, higher accuracy is attainable. This is shown in Fig. 10. The figure also shows the percentage of correctly predicted taken branches which have incorrect target addresses stored in the BTB. As we mentioned earlier, some saving in branch delay is possible since the direction of the branch is correctly predicted, even though the correct target address needs to be restored.

Although we have only shown the results for direct-mapped BTB, other associativities may be considered. For completeness, we show in Fig. 11 the results for il with different associativities for the 512-entry BTB. Note that the accuracy is broken down into two components: "correctly predicted taken branches with correct target addresses" and "correctly predicted not taken branches". For example, as shown in the figure, the two components for il are 47% and 35%, respectively, for the direct-mapped 512-entry BTB implemented without a BPT. These add up to 82%, which is also shown in Fig. 10. Again we have seen considerable improvement in accuracy by the correlation scheme. It is interesting to note that a 4-way BTB gives the best result.

From the simulation we have observed that a set-associative BTB also gives noticeable improvement in accuracy over a direct-mapped BTB for fppp and gcc, but with very minor improvement for the other SPEC benchmarks.

6. Conclusions

In this paper, we have proposed a new dynamic branch prediction scheme which uses the proper subhistory information of a branch to predict the outcome of the branch. The key idea is to relate the subhistory which is being selected for prediction to the most recently executed branches via a shift register. The new scheme is evaluated using traces collected from running the SPEC benchmark suite on an IBM RISC System/6000 machine. It is shown that the proposed new scheme gives considerably higher accuracy than that of the 2-bit counter prediction scheme at the extra hardware cost of one shift register. We have demonstrated that the new scheme is simple and easy to implement. We have also shown that the accuracy of the correlation scheme surpasses that of the 2-bit counter scheme at saturation. When the correlation-based branch prediction scheme is implemented in conjunction with a BTB, more promising results are obtained. The results have shown that the chance for correctly redirecting the instruction fetch after a branch is improved, and the chance for reducing the branch delay is increased, even when a correctly predicted taken branch has an incorrect target address stored in the BTB.

References