In the Proceedings of the 1997 IEEE International Conference on Computer Design (ICCD'97), pp. 82–87, Austin, Texas, October 12-15, 1997.

Elastic History Buffer: A Low-Cost Method to Improve Branch Prediction Accuracy

Maria-Dana Tarlescu and Kevin B. Theobald
School of Computer Science
McGill University
Montréal, Québec, Canada
dana@cs.mcgill.ca, theobald@cs.mcgill.ca

Guang R. Gao
Dept. of Electrical and Computer Engineering
University of Delaware
Newark, Delaware, USA
ggao@capsl.udel.edu

Abstract

Two-level dynamic branch predictors try to predict the outcomes of conditional branches using both a table of state counters associated with specific branch instructions and a buffer of recent branch outcomes to correlate the counters with specific branch histories. However, there is always a question of how much correlation to use, and some programs benefit from higher levels of correlation than others.

This paper presents the Elastic History Buffer *(EHB), a low-cost yet effective scheme that can exploit the property that each branch instruction may have a different degree of correlation with other branches, while keeping the simple structure of a single global branch history. We have simulated the EHB on SPECint92 for two architectures. On average, the EHB has 25% fewer mispredictions than fixed-correlation schemes and 10% fewer than frequency-based branch classification schemes. With limited hardware (1KB), the EHB is close to the optimum measured by repeating the experiments on an "oracle" two-level predictor.*

1 Introduction

Conditional branches are one of the most serious impediments to high performance in modern processors for general applications, and will become an even bigger problem as pipeline depths and issue widths continue to expand. Dynamic branch predictors try to guess the direction of a branch by deriving patterns from previous branch outcomes.[1] "Two-level" predictors [7, 8] look at both the history of branches at the branch instruction in question, and the outcomes of other branch instructions recently encountered. Two-level schemes generally are more effective because the behavior of a branch instruction is often correlated with other recent branches [7].

A typical two-level predictor is shown in Figure 1. This consists of an array of 2^n state counters called the *Pattern History Table* (PHT),[2] and a *Branch History Buffer* that keeps a record of the outcomes of the n most recent branches (in time) regardless of address. To predict an upcoming branch, these n bits are combined with n bits from the address of the branch instruction (typically the least significant) to produce an n-bit index into the PHT. The chosen PHT entry is then used to guess whether the branch is taken or not taken, and the actual branch outcome, once it is known, updates the PHT entry using a next-state function. Most predictors use the 2-bit saturating up-down counter [5] shown in the figure, in which the left state bit makes the prediction (1 = taken). Common combining functions include gshare [6], in which the index is the exclusive-OR of the two n-bit inputs, and GAs [8], which concatenates the k most-recent bits of the shift register with $(n - k)$ bits from the PC address.

Predictors based on the latter technique use a fixed value of k, i.e., a constant degree of correlation between a branch instruction and neighboring branches. However, the optimal degree of correlation may vary from branch to branch, and a fixed-correlation scheme may have poor results when a particular branch instruction has more or less correlation than assumed. Chang et al. showed that the accuracy of a simple two-level branch predictor tends to decrease with increasing BHB length for branch instructions that are mostly taken in one direction (e.g., more than 90% of the time) [1].

If a large value of k is used, then one branch instruction can be mapped to many different entries in the PHT. If that instruction's behavior is independent of branches at other addresses, then a large k will hurt performance in two ways. First, the large number of PHT entries per branch

[1] In this paper, "branch instruction" refers to a conditional branch op at one specific address in the code, while a "branch" is one *occurrence* of a branch instruction.

[2] This term is from the taxonomy paper of Yeh and Patt [8], though other names also appear in the literature.

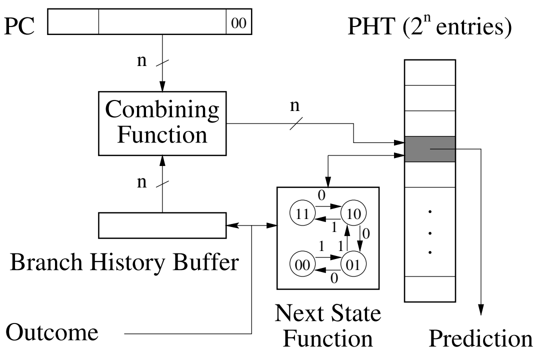

Figure 1. Two-level branch prediction

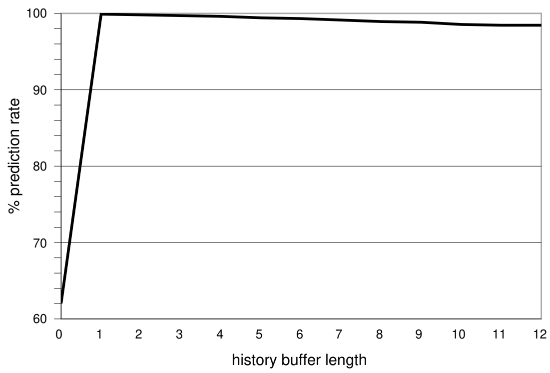

Figure 2. A weakly-correlated branch

instruction will slow the predictor's ability to adapt to dynamic changes in branch behavior. Second, since there are many entries for one branch instruction, there will be many more collisions with other branches, which will usually lead to more mispredictions.

Chang et al. showed that branch prediction rates could be improved by detecting mostly-one-way branch instruction using profiling, and then having them use fewer bits from the BHB, so that they create less interference in the PHT. However, we believe further gains in performance can be attained, both by increasing the number of correlation options available to a branch, and by basing this choice on observed dynamic branch behavior rather than simple frequency.

This paper presents the *Elastic History Buffer* (EHB), which addresses this problem by allowing the amount of correlation to be specified for each particular branch instruction in the code. Correlation is determined by the number of bits from a global Branch History Buffer to use when making predictions. This number is encoded in the branch instruction opcode and is derived from branch characteristics provided by a preliminary "profiling" run. Only minimal modifications to an existing simple correlation-based system are required.

We have performed simulations which demonstrate that for small (1KByte) branch prediction tables comparable to those used in current processors:

- The EHB has an average of 25% fewer mispredictions than fixed-length correlation schemes, and 10% fewer mispredictions than a frequency-based classification system.
- The EHB performs almost as well as an ideal EHB with infinite storage (within 10% in terms of mispredictions).

Furthermore, we have analyzed individual branches in our benchmarks to learn more about branch behavior. The observations we report, which originally led to the design of the EHB, provide a good intuitive understanding of why the EHB does well.

The remainder of this paper is divided as follows: The next section describes the EHB in detail, beginning with a conceptual view of the system and ending with discussions of hardware implementation costs and algorithms for profiling. Section 3 covers the experiments performed in this study and their results. The final section gives our conclusions and discusses future work.

2 The Elastic History Buffer

This section describes the Elastic History Buffer (EHB) branch prediction scheme. The first part describes the general concept, and discusses implementation issues. The second part explains how the profiling phase collects information on branch characteristics. The final section describes how profiler results are used to determine the best way for each branch instruction to use the dynamic prediction hardware.

2.1 Basic concept

The Elastic History Buffer (EHB) is based on observations we made of the behavior of branches in selected benchmarks. We have observed that the relation between branch frequency and ideal history buffer length does not hold for all branches. For instance, Figure 2 shows the prediction rate as a function of BHB length for a particular branch instruction in gcc. This instruction is not "mostly-one-direction," yet is still negatively affected by an increase in history buffer length.

In general, we have seen that there is an optimal amount of correlation for each branch, and that this amount is not always related to branch frequency. The EHB scheme allows an arbitrary number of bits from the BHB to be used,

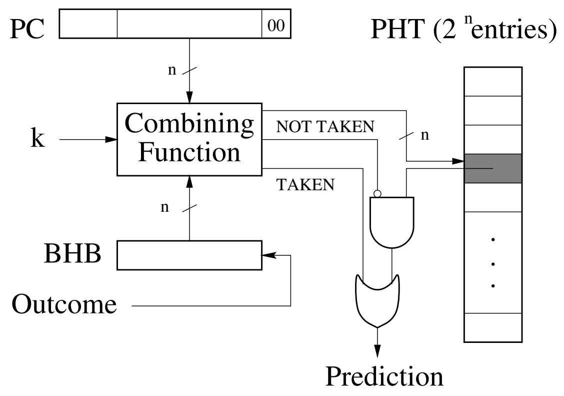

Figure 3. The Elastic History Buffer

allowing finer granularity in the classification of branches. Furthermore, branch classification is done according to each branch instruction's actual performance in a dynamic predictor, rather than just its raw frequency.

The general scheme is illustrated in Figure 3. The logic is similar to the GAs predictor described in Section 1, except that k is no longer fixed. Instead, k is encoded into the branch instruction itself. This value is fed as a control input to special shifting and masking circuits, which form an index into the PHT by concatenating the lower $n - k$ bits of the branch instruction address with the lower k bits of the BHB (the k most recent branches). The next-state function (not shown) is identical to Figure 1.

The control input should have enough bits for k to take any value between 0 to n, inclusive. Two additional patterns are included in this encoding, to enable the predictor to bypass the PHT and assert a prediction of either *taken* or *not-taken*. When one of these is used, the PHT is not updated. This allows some branches to be predicted statically, which may relieve pressure on the PHT and reduce collisions.

We thus place branch instructions in different classes. Branches that use dynamic prediction with k bits from the BHB are in class d_k. Branches that bypass the PHT and rely on static prediction are in a separate class s_0, with subclasses s_t (normally taken) and s_n (normally not-taken).

The EHB allows the right amount of correlation to be used for each specific branch instruction. Instructions whose behavior is independent of the surrounding branches can be given a small length value (e.g., d_0), so that their entries in the PHT "warm up" quickly and are less prone to interfere with other branches. Instructions that are highly correlated can be placed in the upper classes (as high as d_n), while others can belong to intermediate classes. Finally, branches which are both strongly biased *and* uncorrelated may dispense with dynamic prediction entirely.

The EHB scheme has the disadvantage that some bits from the branch instruction must be allocated for the encoding of k. But this is not a significant shortcoming. Four bits would be enough to encode all classes from d_0 to d_{13} (corresponding to a 2KByte PHT), with two extra patterns for s_t and s_n. This would reduce, by a factor of 16, the range of addresses that could be reached by a conditional branch. However, since distant branches are rare, they don't have a pressing need for accurate branch prediction. Some processors already have separate opcodes for branches likely to be taken and those likely not to be taken. One of these opcodes could be used for branch instructions with a class field, leaving the other for branches that require a full-length branch offset. Branches of the latter type could still take advantage of two-level branch prediction, but they would have to use a default value of k, like the fixed-buffer schemes.

Hardware costs are another factor to consider when comparing branch prediction schemes. The hardware needed to implement the control logic is only marginally more complex than the simple GAs predictor. Both require a shift register for the Branch History Buffer. In addition, the EHB would require a barrel-shifter to move the PC address bits to the proper alignment with the BHB bits, according to the value of k, and some masking circuits. No additional storage would be required, as with more ambitious designs such as SAs [8]. Furthermore, the small-scale shifting and masking operations should be faster than the decoding and memory access operations required by those other schemes.

2.2 Branch profiling

Because class values must be encoded in each branch instruction, a profiling phase is necessary to determine these classes. The profiler determines how many bits are appropriate by testing the branches with different history lengths to determine the best length for each. The profiling is performed as follows (assume a 2^n-entry PHT and a maximum BHB length of n):

1. Create $(n + 1)$ 2^n-entry Pattern History Tables (PHT_0 to PHT_n) and an n-bit BHB.

2. For each branch instruction b, allocate and initialize an array of $(n + 3)$ counters:

   - T_b for the number of times b is taken;
   - N_b for the number of times b is not taken;
   - A counter $P_{b,k}$ for each PHT_k $(0 \leq k \leq n)$.

3. Execute the code. For each occurrence of b, update the counters associated with b:

   - Increment T_b if the branch is taken, or N_b if the branch is not taken.

- For each PHT_k ($0 \leq k \leq n$), operate it as a fixed-length GAs. Concatenate the lowest $(n-k)$ bits of the address of b with the lowest k bits of the BHB, access PHT_k with this index, and predict accordingly. If the prediction is successful, increment $P_{b,k}$. Update the chosen entry in PHT_k.
- Finally, shift the branch outcome into the BHB.

Profiling is already used with many advanced compilers to allow programmers to get more performance from their code. There is one problem with standard profiling techniques which must be addressed. Profiling compilers generally add extra instructions to compiled code in order to generate the profile. However, these extra instructions will move branch instructions to different addresses, which will move them to different areas of the PHT, possibly causing the interferences so generated to be different from those in the final run. To get around this problem, the compiler should create a mapping from *profiler* addresses to *final* addresses, and to convert addresses from the former to the latter each time step 3 in the algorithm above is applied to a particular branch.

2.3 Branch classification

After the profile is complete, the branch instructions must be classified. For each instruction b, find the maximum of $\{T_b, N_b, P_{b,0}, \ldots P_{b,n}\}$ and choose the lowest class which produces this maximum. (Class A is "lower" than class B if branches in class A map to potentially fewer locations in the PHT than those in class B; s_0 is "lower" than d_0, and d_k is "lower" than d_{k+1}.) This class can then be encoded and added to instruction b of the final version of the code.

There are several weaknesses with the EHB classification method. First, the profiling state attempts to select the class that is most appropriate to a given branch instruction by simulating its behavior in a PHT. However, the simulation assumes that all branches using a specific PHT are in the same dynamic class (use the same number of bits from the BHB). After classification, branches will be in many different classes, and the interferences that affected the profiling stage may be different when the final code is run. Thus, the classes derived by the algorithm above will only be approximate. However, because each branch instruction can affect all other branches, computing the most optimal k for each branch would be computationally infeasible. Nevertheless, our "heuristic" approach produces results very close to the ideal limit, as we show in the next section.

Another problem with EHB classification, as with all static profiling techniques, is the fact that the inputs to the profiling run of the program will probably be different from the inputs to real executions. This issue will be discussed in the next section.

Bench-mark	Input	Ops $\times 10^6$		Branches $\times 10^6$	
		Sparc	i860	Sparc	i860
gcc	cexp.i	14.3	23.3	2.2	3.3
espresso	opa.in	149.3	126.9	24.6	22.0
li	(queens 8)	967.1	1,248.8	156.7	168.2
sc	load1	90.7	—	13.5	—
compress	in (ref)	89.0	112.5	11.7	15.1
eqntott	int_pri_3	1,124.0	1,393.0	229.4	352.4

Table 1. Benchmarks Used

3 Experimental results

To measure the effectiveness of the Elastic History Buffer and to compare it to other methods, we built two different testbeds for our experiments. The first tool is for programs running on Sparc processors. The Spy trace generator [4] runs a Sparc executable file and feeds a dynamic instruction trace into our program, which can simulate either the profiling stage or the final execution of code. Classification of branches (based on output of the profiler) is done by a separate program. The second tool is for programs running on i860 processors, and is based on the SEMi i860 simulator [3] augmented with routines to simulate the profiling and final stages of the branch predictor. Branch classification is performed at the beginning of the final stage. Neither tool can trace inside system calls. The testbeds were implemented independently to minimize the risk of undetected correlated bugs.

The C benchmarks from SPECint92 were used (see Table 1). Each was compiled for the Sparc using gcc 2.6.3, and for the i860 using the PGI C compiler 3.1.2. The preprocessing phase of eqntott was bypassed, since neither tool can handle forks. Similarly, only the main compiler (cc1) of gcc was used. The sc benchmark could not run on the i860 simulator because it relies on non-standard libraries not available in our i860 system. Overall averages use arithmetic means rather than weighted means as the former are considered more appropriate for this type of study [2].

Profiling was performed as described in Section 2.2. The profiles were used to classify branch instructions as described in Section 2.3. A final run was performed to measure the prediction rate of the classified branches using the simulated EHB.

In an attempt to see how close to "ideal" a realistic EHB with interference can perform, we ran an additional profiler called an "oracle" profiler. The oracle's data collection method is similar to Section 2.2, but instead of having all branch instructions share a single PHT for each value of k, each branch instruction has its own private set of $(n+1)$ tables. The results for this profiler are then used

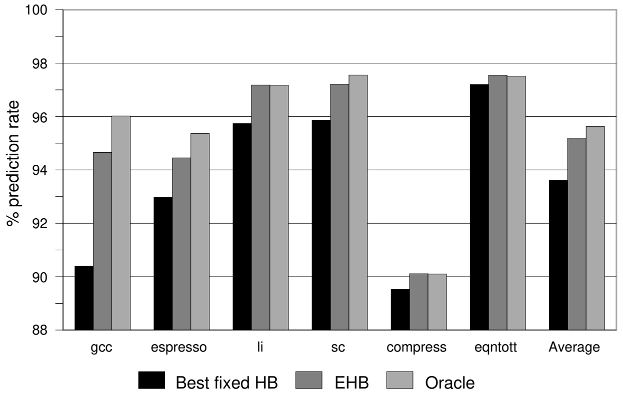

Figure 4. EHB vs. GAs and Oracle (Sparc)

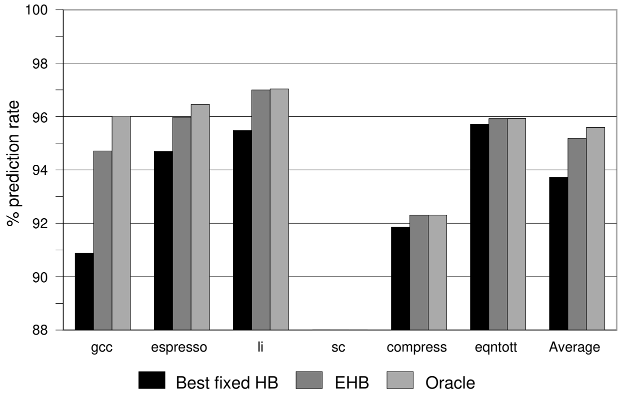

Figure 5. EHB vs. GAs and Oracle (i860)

to classify branches as described in Section 2.3. Since this model assumes no interference, it is not necessary to do a final run; the total number of successfully-predicted branches is simply the sum of $\max(T_b, N_b, P_{b,0}, \ldots P_{b,n})$ over all b. The result measures the benefits of branch correlation, optimized on a per-branch basis, while ignoring the effects of branch interference. It therefore gives an upper bound on the performance that can be expected by a two-level correlation-based scheme for a given maximum BHB length.

3.1 Effectiveness of the EHB

Figures 4 and 5 compare the performances, for each architecture, of the GAs with a fixed History Buffer (FHB), the EHB, and the interference-free oracle described in the previous subsection. The first bar in each group shows the prediction rate of the *best* FHB predictor. The FHB figures are optimistic because the BHB length k is different for each benchmark. The choice of k can affect the prediction rates considerably for some benchmarks (more than 2% in our experiments). Even with the best FHB for each benchmark, EHB has a 1.8% better prediction rate, which translates into a 25% reduction in mispredictions.

According to our observations, some types of programs (such as gcc and espresso) benefit more from the EHB than others (compress, eqntott). The reason for this difference is two-fold: on one hand, benchmarks such as gcc and espresso have a large number of static branches, which means that in an FHB scheme, branch interference is very frequent and its negative effect on prediction accuracy is important. For these benchmarks, removing branches which can be predicted statically from the competition for BPT entries reduces interference and thus increases overall accuracy. On the other hand, in programs such as eqntott, which have a very regular structure, branches have often very similar behaviors (most of them are loop branches with numbers of iterations proportional to the problem size). For these programs, the EHB scheme cannot bring very much improvement over an FHB scheme, because there is not enough variety in the correlation depths of the different branches that can be exploited.

The last bar in each group shows the performance achieved by the oracle profiler. We can see that, on average, the prediction accuracy of our scheme is within 0.5% of the upper bound (10% of mispredictions). It is important to notice that there are some benchmarks (compress, eqntott) for which the accuracy of the EHB scheme practically reaches the upper bound. Moreover, we notice that these benchmarks are exactly the ones for which we reported the smallest improvements over the fixed-length schemes. This confirms our supposition that the reason those benchmarks don't show a higher benefit from the use of our scheme is related to an actual lack of correlation between their branches.

We also ran simulations on SAs [8] and gshare [6] predictors of the same size (1KBytes of storage). For this amount of hardware, neither SAs nor gshare predicted as well as the FHB for any of the benchmarks. We omit this data for space reasons.

Chang et al. proposed using static profiling to classify branches according to how often they take their preferred direction, and then to use short histories to predict branches that heavily favor one direction, and long histories for branches that are less certain [1]. The EHB provides more than just two classes for branches, and classifies them according to their performance in the profiling stage, rather than just their frequency of direction. To see if this extra complexity is justified, we added a simulator of their system to our Sparc tool and ran our benchmarks with their branch simulator. The results are shown in Figure 6. They show that the finer granularity of correlation and the more complex profiling scheme of EHB do provide a tangible benefit

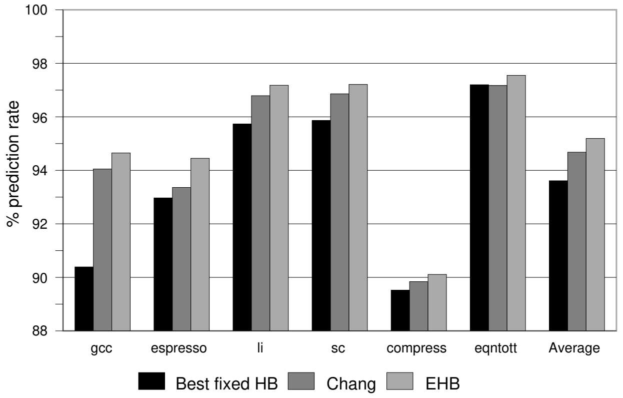

Figure 6. FHB, Chang and EHB (Sparc)

(mispredictions reduced by 10%).

3.2 Robustness of the EHB

All static profiling methods suffer from the shortcoming that the behavior of the code during the profiling stage may not accurately represent the behavior of the code for a real input case. We are currently researching training strategies for the EHB. A full treatment of this topic is beyond the scope of this paper. However, we have found that prediction rates vary from the self-training case by typically less than 0.5%, provided a suitable training input (that exercises all branches) is used. This means EHB is still better than the frequency-based classification even when the latter trains on its own input.

4 Conclusions and future work

In this paper, we have presented the Elastic History Buffer, a mechanism for improving branch prediction by specifying the amounts of branch history used by each branch instruction. We began with the observation that branch instructions have widely varying amounts of correlation with other recent branches, and we further claimed that this relation cannot be determined merely by looking at the frequency characteristics of a branch. This claims was supported with examples drawn from SPEC benchmarks.

The EHB has a simple and inexpensive hardware structure similar to a GAs, but allows each branch to specify how many bits to use from the Branch History Buffer, with the remaining bits coming from the branch address itself. The bit count is encoded into a small field in each branch instruction. A branch profile is obtained by running the application and simulating the branches' behavior on different fixed-HB systems, then using the statistics collected to choose the best setting for each branch instruction.

Simulations performed on two different architectures confirm that the EHB predicts branches more accurately than either a fixed-HB system or a branch predictor based on classifying branches according to frequencies alone. The EHB results for most benchmarks are quite close to the theoretical limits for a two-level correlation-based system, although there may be room for improvement by expanding the prediction hardware.

Up to now, we have only studied single-predictor systems, because we wanted to show that EHB is practical for small branch prediction circuits in current microprocessors. Since hybrid dynamic systems employing multiple predictors are now getting more attention, one interesting topic for further research would be to test the efficacy of EHB as one component of a multi-predictor system. Another place to look for possible improvements is the manner in which indices into the PHT are constructed; other addressing schemes based on the EHB principle may yield further gains. Finally, we would like to study the issue of training more thoroughly, in order to develop profiling strategies that produce better and more consistent classifications.

Acknowledgment

The authors gratefully acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) of Canada and the National Science Foundation (NSF) of the United States.

References

[1] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. Patt. Branch classification: A new mechanism for improving branch predictor performance. In *Proc. of MICRO-27*, pages 22–31, San Jose, Calif., Nov.–Dec. 1994.

[2] B. Fagin and A. Mital. The performance of counter- and correlation-based schemes for branch target buffers. *IEEE Trans. on Computers*, 44(12):1383–1393, Dec. 1995.

[3] H. H. J. Hum, O. Maquelin, K. B. Theobald, X. Tian, G. R. Gao, and L. J. Hendren. A study of the EARTH-MANNA multithreaded system. *Intl. J. of Parallel Programming*, 24(4):319–347, Aug. 1996.

[4] G. Irlam. Spa. World-Wide Web page URL: http://www.base.com/gordoni/spa.html.

[5] J. K. F. Lee and A. J. Smith. Branch prediction strategies and branch target buffer design. *Computer*, 17(1):6–22, Jan. 1984.

[6] S. McFarling. Combining branch predictors. WRL Tech. Note TN-36, Digital Equipment Corporation, Jun. 1993.

[7] S.-T. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proc. of ASPLOS-V*, pages 76–84, Boston, Mass., Oct. 1992.

[8] T.-Y. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *Proc. of ISCA-20*, pages 257–266, San Diego, Calif., May 1993.