Improving Branch Prediction Accuracy in Embedded Processors in the Presence of Context Switches

Sudeep Pasricha, Alex Veidenbaum
Center for Embedded Computer Systems
University of California, Irvine, CA 92697, USA
{sudeep, alexv}@cecs.uci.edu

Abstract

Embedded processors like Intel's XScale use dynamic branch prediction to improve performance. Due to the presence of context switches, the accuracy of these predictors is reduced because they end up storing prediction histories for several processes. This paper shows that the loss in accuracy can be significant and depends on predictor type and size. Several new schemes are proposed to save and restore the predictor state on context switches in order to improve prediction accuracy. The schemes differ in the amount of information they save and vary in their accuracy improvement. It is shown that even for a small 128 entry skew predictor, 2 - 6% improvement in prediction rate can be achieved (for an average context interval of 100K instructions) for different embedded applications while saving and restoring a minimal amount of state information (less than 32bits) on a context switch.

1. Introduction

Modern embedded processors use pipelining to exploit parallelism and improve performance. Conditional branches in the instruction stream degrade performance by causing pipeline flushes. Branch prediction mechanisms can overcome this limitation by predicting the outcome of the branch before its condition is resolved. As a result, instruction fetch is not interrupted as often and the window of instructions over which ILP can be exposed increases. In fact, accurate branch predictors can eliminate over 90% [13] of these pipeline stalls and are thus critical to realizing the performance potential of a processor. Improving branch prediction accuracy is important because the new generation of embedded processors have deeper pipelines, which result in larger misprediction penalties. In the XScale [8] processor which has a 7 stage pipeline, the penalty for each misprediction is as high as 4 cycles.

Most processors use dynamic branch prediction [5,6,12,13,15] to predict branch directions. Dynamic predictors record and utilize information from previous runs of a static branch instruction to predict its outcome in the future. This requires additional hardware to store the branch history. These predictors dynamically adjust their prediction to match the changing behavior of a branch instruction as the program executes.

One aspect of branch prediction that has largely been ignored is the effect of context switches. In typical systems, several processes are in the active queue at any given time and they share the branch predictor structure. Each process runs for its allotted time slice and then yields the processor to allow another waiting process to execute. Unless steps are taken to change the state of the predictor structure, it will contain stale information from the run of the previous process when the new process commences execution. Since different processes generally have completely different branch behaviors, reusing the stale information will increase the misprediction rate. It is desirable to overcome this limitation. This research explores the effects of context switches on dynamic branch prediction schemes, in the context of embedded processors which have stringent hardware resource constraints. We find that context switches cause substantial decrease in prediction rate – an average of 4% for the skew predictor for instance. Having established that context switches degrade prediction accuracy, we then propose several methods to alleviate this performance loss for the skew scheme, at different costs to the architect. We choose the skew predictor because it provides the best performance for a limited hardware budget, but the proposed schemes work for other dynamic predictors too.

This paper is organized as follows: Section 2 presents previous work in this area. Section 3 illustrates the effect of context switches on these dynamic branch predictors. Section 4 presents schemes to improve branch predictor performance in the presence of context switches. Section 5 reports the simulation results and our analysis for the proposed schemes. Section 6 provides some concluding remarks.

2. Related Work

Several papers on branch prediction acknowledge the effects of context switching on branch prediction accuracy.

Yeh and Patt [6] examined the effect of context switches on two-level branch prediction schemes. They found that the average accuracy degradations for the PAp, PAg and GAg schemes are less than one percent for a context switch interval of 500K instructions. However in their experiments they did not change the pattern history table on a context switch, which explains the exceptionally small decrease in prediction accuracy for the large predictor structures used. In an actual multi-programming environment, the pattern history tables for different processes will differ and if the PHT is kept unchanged, prediction accuracy will suffer.

Gloy, et al [7] studied dynamic branch prediction schemes on system workloads. They found that including kernel level branches with user level branches in experiments significantly affected branch prediction accuracies, increasing aliasing and thus decreasing prediction accuracy. They emphasized the need to consider the whole system rather than just user level code when evaluating branch prediction schemes. This motivated us to study the impact of context switches on the performance of dynamic prediction schemes.

More recently, Michele Co. and K. Skadron [1] claimed that context switching has negligible effect on branch predictor performance. They measured the context switch interval based on the default time slice value for Windows NT (25 ms), and it turned out to be around 50M instructions. Our experiments calculate this interval from context switching information obtained from several multi-programmed systems with varying workloads. The interval we obtain is much smaller than theirs and agrees with the findings of previous studies [2, 3, 4, 10] that consider the effect of context switching on branch prediction.

A.S. Dhodapkar and James E. Smith [3] presented the case for saving predictor information on a context switch. They proposed (for a gshare predictor) saving the most significant bits of all the counters in the branch predictor tables on a context switch. They also proposed setting the predictor counters to weakly taken on a context switch, as an alternative. This work is the only one that we know of which proposes mechanisms to improve predictor accuracy in the presence of context switches. They rightly identify the need to reduce the 'learning time' of the predictor after a context switch by restoring previously saved prediction values into the predictor. This paper extends this study and proposes several other mechanisms that will improve performance.

3. Effect of Context Switches

Context switches can occur during program execution for several reasons such as I/O requests, system calls, page faults, expiration of time slice etc. The frequency of these context switches depends on factors like the number of applications active on a system, the types of these applications, the operating system used and the scheduling scheme. We performed experiments to determine context switch intervals on several different systems, with varying workloads and running different operating systems such as UNIX, Linux and Windows 2K. For UNIX and Linux we used the *vmstat* utility while for Windows we used the *ntimer* utility, which is part of the Windows 2000 Resource Kit. Our results indicate a context switch frequency varying from 100/sec to 8000/sec and a context switch interval ranging from 75K – 1000K instructions. For instance, one of the machines we tested was a SUN UltraSparc-II workstation running SunOS 5.8 at a maximum clock speed of 400 MHz. The context switch frequency on it varied from 400/sec to 4500/sec, with a changing workload. If it is assumed that one instruction is executed every cycle, we get a context switch interval of 90K instructions for the higher end. In another experiment, we tested a 996 MHz Intel PIII machine running Windows 2000 and found that the context switch frequency varied from about 1000/sec to 8000/sec which gives a context switch interval of around 125K instructions for the higher end, assuming an IPC of 1. These numbers are in line with the results obtained in [3] as well as other studies done previously that analyze branch predictor performance in the presence of context switches [2, 4, 10].

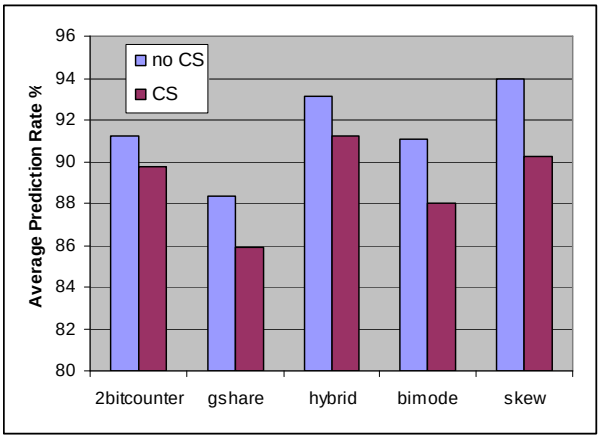


Figure 1. Predictor accuracy in the presence of context switches (128 entries, 100K CS interval)

Figure 1 shows the average performance degradation for several commonly used dynamic branch prediction schemes in the presence of context switches, for around 20 benchmarks from the MiBench [11] suite. "no CS" represents the ideal case when no context switches occur. We do not modify the branch predictor tables on a context switch, allowing the prediction information of

different processes to overlap. Due to the destructive aliasing from overlapping processes, performance deteriorates in all cases. We simulate this in the "CS" case by filling the predictor tables with spurious values (inverting all the bits in some cases and inserting random values in others) at the point when a context switch is scheduled to occur. We chose to compare predictors with a small size of 128 entries because the XScale processor uses a 128 entry bimodal predictor and because it is typical for embedded processors to have small predictor sizes. Results are shown for the intervals of 100K, which gives a lower bound of the performance for the predictors, in the presence of context switches.

The first point to note from the figure is that the prediction accuracy of certain predictors like skew and hybrid is more than that of the simple 2bit counter (bimodal) which is used in XScale. This improvement comes at the cost of additional tables in these predictors. But future generations of embedded processors are expected to have deeper pipelines to exploit parallelism and consequently larger predictor sizes to improve prediction rates, since misprediction penalty with deeper pipelines will be more. The main observation from Figure 1 however is that the loss in prediction accuracy is significant for all of the predictors, due to the presence of context switches. Due to lack of space, the performance for individual benchmarks cannot be shown, but the difference in prediction accuracy varies from 4-7% for applications such as jpeg, rijndael, gsm, basicmath, fft and qsort from the MiBench [11] suite. This shows how important it is to address the effect of context switches on prediction accuracy.

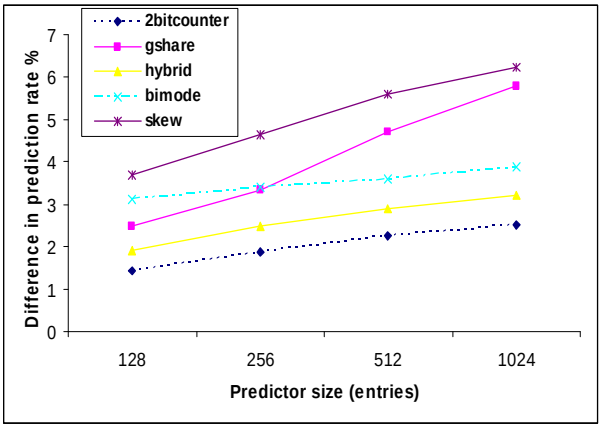


Figure 2. Difference in prediction rate due to context switches for dynamic predictors with varying hardware budgets

Figure 2 shows the difference in prediction accuracy for the cases when context switches are absent and when they are present, for different predictor sizes – 128, 256, 512, 1024 entries in the predictor tables. Although increasing predictor size improves performance because larger predictor tables result in lesser destructive aliasing, it is apparent from the figure that the effect of context switches becomes more prominent with increasing predictor size.

Other experiments (not shown here) show that with an increase in context switch interval, performance for all the predictors improves. Nonetheless, low context switch intervals are quite frequently encountered in systems, and for these intervals the degradation in performance is large (Figure 1). In the next section mechanisms to overcome this performance loss are proposed.

4. New Schemes to Improve Accuracy

On a context switch, the predictor structure contains information for the process that just finished execution. This information does not accurately represent predictions for other processes, as was shown in Figure 2. One existing scheme is to flush the predictor bits to zero every time a context switch occurs [8]. Another scheme [3] sets all the predictor table entries to weakly taken. This section describes new schemes we use to improve on the above.

If there were a way to save the entire prediction information from the predictor tables for a process, and then restore the predictions into the predictor structure when the process resumes execution, there would be no loss in performance. However, saving and restoring entire predictor structures as was done in [3] can be prohibitive both in terms of time and memory space. Therefore we investigate several schemes for saving "representative" portions of the prediction information. For instance, one simple scheme we propose saves 1 bit per table for each process when a context switch occurs. This bit is the predominant bias of the predictor tables – either taken or not taken. When the process resumes execution, the bit is used to bias the counters in the predictor tables based on its value. We call this the *majority bias* scheme.

Another scheme "compresses" the predictor state and saves N bits per process. The value of N is selected so as to achieve a desired accuracy improvement without large overhead. A compression algorithm partitions a predictor table into blocks of *k* entries and stores the state information for each block. The state can be the dominant bias bit for a block. Alternatively, 2 bits of state can be saved per block (the scheme evaluated in this paper). On a context switch, the number of entries in the block set to strongly not-taken, weakly not-taken, weakly taken and strongly taken is used to save the state of a block as following:

- 00: if there are more strongly not-taken entries than strongly taken entries in the block. If these are equal then the weakly not-taken and weakly taken entries

are compared and a 00 is saved if there are more weakly not-taken entries.

- 01: if there are more strongly taken entries than strongly not-taken entries. If these are equal then the weakly taken and weakly not-taken entries are compared and a 01 is saved if there are more weakly taken entries.
- 10: otherwise if the overall number of taken and not-taken entries is the same, we save a 10.

When the process is resumed and before it commences execution, the two saved bits for each block of the predictor table are used to restore the state as follows: If 00 was stored, we bias all the counters in the block to weakly not-taken. If the saved value was 01, we bias all the counters to weakly taken. For the case of a 10, we bias successive counters in the tables alternately to weakly taken and not-taken. To implement this scheme, hardware counters are used together with some combinational logic to route and store the data.

Another way to reduce the amount of predictor information saved would be to save just the most significant bit of the two bit counters in the tables as proposed in [3]. This *snapshot* of the most significant bits of the counters in the tables reduces the information stored by half. Instead of saving snapshots for all the tables, a *partial snapshot* can be taken for a subset of the tables in the predictor. This further reduces the information stored, at the cost of reduced prediction accuracy.

To save and restore the predictor information, we propose the use of an L2 cache, bypassing the smaller L1 caches, or alternately we can also use a small dedicated buffer for the purpose. The results in the next section show that schemes which require saving as little as 19 bits for a 128 entry predictor can achieve a significant improvement in prediction rate (up to 6%) for the skew predictor which has the best prediction rate in the absence of context switches compared to the other dynamic predictors selected for study. The time penalty for doing this is a few extra cycles for saving and restoring the information, including the combinational logic delay for compression. This is a small price to pay when compared with the large improvement in performance gained from more accurate branch prediction. It is also very small compared to the overall context switch overhead and can be done entirely in hardware. Recall that the minimum penalty on a branch misprediction for the XScale processor is 4 cycles. Thus if a scheme exhibits even a marginal improvement in prediction rate, it can justify the overhead of saving and restoring the predictor information.

5. Performance Evaluation

Our goal was to reduce the prediction accuracy loss due to context switches for dynamic branch predictors. The skew branch predictor [12] (Figure 3) was selected because it gave the best performance among the 128 entry predictors considered (Figure 1).

All simulations were performed on a modified version of the *sim-outorder* simulator from SimpleScalar [19] version 3.0. The processor modeled uses a configuration similar to the XScale processor: 32KB data and instruction L1 caches with 32 byte lines and 1 cycle latency, no L2 cache and 50 cycle main memory access latency. The machine is in-order and a 32 entry load/store queue, with an issue width of 2. It has one integer unit, one floating point unit and one multiply/divide unit. The branch predictor has 128 entries, and the instruction and data TLBs are 32 entry and fully associative.

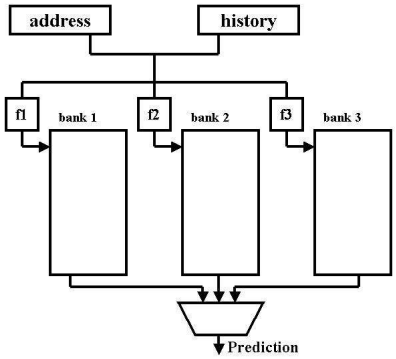


Figure 3. skew predictor

We selected benchmarks from the MiBench [11] suite which is designed to be representative of several embedded systems domains. Due to lack of space, we show results for a limited set of benchmarks – basicmath (automotive and control), ghostscript, ispell (office automation), rijndael (security), fft and gsm (telecommunications). A context switch interval of 100K instructions was chosen to represent a lower bound on performance when context switches are present. A small 128 entry skew predictor was used. All simulations were run till termination.

5.1. Results

Figure 4 shows the results for the proposed schemes. “zero” refers to scheme which sets all entries to zero on a context switch while “taken” sets the counters to weakly taken. For all subsequent schemes that save predictor state we assume that the 7 bit global history register is also saved. “snapshot 1”, “snapshot 2” and “snapshot 3” take *partial snapshots* of 1, 2 and all 3 of the tables in the skew predictor saving 128 bits per table (128+7, 256+7 and 384+7 bits overall, respectively). “bias32” uses the block bias scheme and saves 8 bits per table (24+7 bits

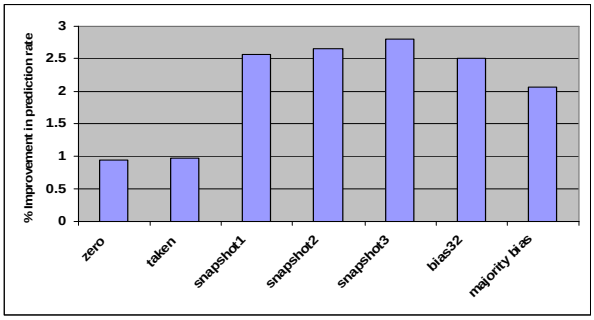
overall) as explained earlier while "majority bias" is the simple scheme that saves just 1 bit per table (3+7 bits overall).

From the figures it is apparent that the flush to *zero*, and *taken* schemes are outperformed by schemes which save predictor state. As expected, the *snapshot* schemes store a lot of information and performance improves when more tables are considered. The *bias32* scheme interestingly performs nearly as well as the *snapshot1* scheme but saves just 24+7 bits instead of 128+7. The *majority bias* scheme performs a little worse than *bias32* overall (gsm is an exception) but saves just 3+7 bits overall! For gsm, the majority bias scheme actually performs better than the other elaborate schemes because a small predictor size of 128 entries results in a lot of aliasing in this case. As a result, resetting the counter bits according to the table bias on a context switch actually improves performance more than restoring the counters closer to their original state with the other schemes.

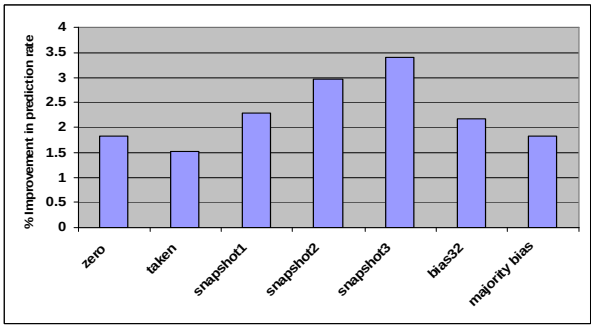
Overall, the bias32 and majority bias schemes are better than the negligible overhead schemes (*zero*, *taken*) providing good prediction accuracy without saving as much information as in the *snapshot* schemes.

6. Summary and Conclusion

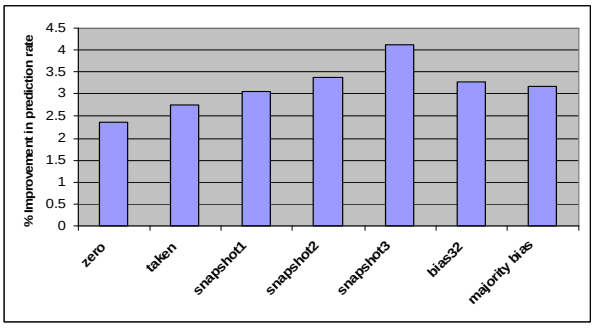
This paper evaluated the loss of prediction accuracy in the presence of context switches for several branch predictors. The loss of accuracy was shown to be significant for all the considered predictors.



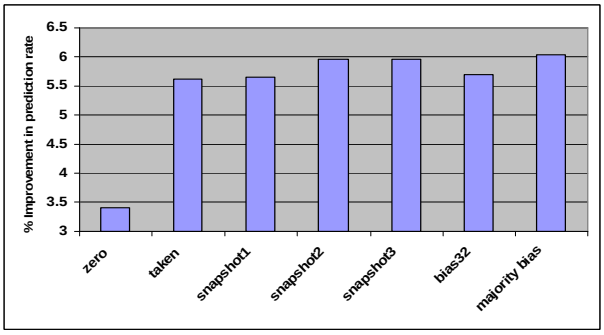
(a) adpcm



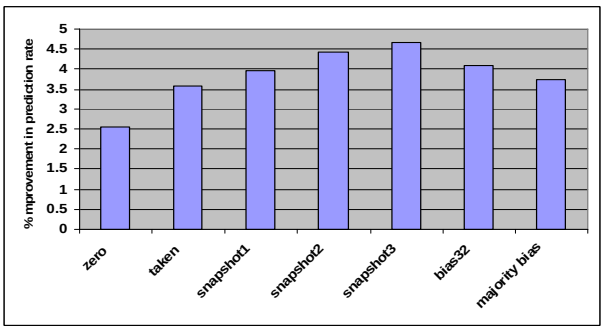
(b) ghostscript



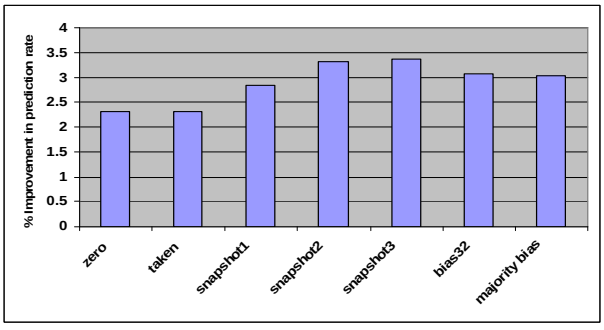
(b) fft



(c) gsm



(d) basicmath



(f) ispell

Figure 4. Improvement in prediction rates for the proposed schemes, in the presence of context switches (128 entry skew predictor, 100K CS interval)

Several new mechanisms to restore the lost accuracy for the skew predictor were presented. The skew predictor was selected due to its high accuracy despite its small hardware budget (128 entry) and the new schemes evaluated assuming a 100K instructions context switch interval. The latter is a lower bound on observed context switch interval size.

Most of the proposed schemes involve saving and restoring varying amounts of predictor information on a context switch. This entails an overhead which needs to be considered in selecting a practical scheme. It was shown that for the *bias32* scheme, saving less than 32 bits can improve the prediction rate from 2 - 6%. The *majority bias* scheme performs slightly worse than the bias32 scheme on the average, but saves just 10 bits to achieve improvement. Both these schemes perform significantly better then the previously-proposed low-overhead schemes - flush to *zero* and weakly *taken*. Overall, we do not consider the snapshot schemes practical to implement or their additional performance improvement worthwhile. This paper concentrates on prediction accuracy and does not evaluate the impact on overall CPU performance. Thus we do not describe the design of the new schemes or their latency in much detail. We believe that it can be done efficiently for the low overhead schemes where the information to be stored is continuously generated from the predictor data by dedicated logic. Therefore only saving/restoring the compressed information takes time. This may be doable purely in hardware without executing additional instructions.

These proposed mechanisms are not limited to the skew scheme and can be used effectively with other dynamic prediction schemes. Similar performance improvements can be expected for the hybrid, bimode, and gshare predictors. In fact, any dynamic predictor using a bimodal table structure can benefit from our schemes. Schemes that involve large pattern history tables such as the alloy predictor are harder to deal with when it comes to reducing the information saved and restored on a context switch. This remains subject of future research.

7. References

[1] Michele Co., K. Skadron "The Effects of Context Switching on Branch Predictor Performance". *Proceedings of the 2001 ISPASS*, November, 2001, Tuscon, AZ

[2] Marius Evers, Po-Yung Chang, Yale N. Patt "Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches". *Proceedings of the 23rd ISCA*, pp. 3-11, 1996.

[3] Ashutosh S. Dhodapkar and James E. Smith "Saving and Restoring Implementation Contexts with co-Designed Virtual Machines". *Workshop on Complexity-Effective Design*, June 30 2001, Goteborg, Sweden.

[4] R. Nair. "Dynamic Path-Based Branch Correlation". *28th MICRO*, pages 15--23, November 1995.

[5] A. N. Eden and T. Mudge, "The YAGS Branch Prediction Scheme", *Proceedings of the 31st Annual ACM/IEEE MICRO*, pages 69-77, 1998.

[6] Tse-Yu Yeh, Yale N. Patt, "Alternative Implementations of Two-Level Adaptive Branch Prediction" *Nineteenth ISCA*, 1992

[7] Nicolas Gloy, Cliff Young, J. Bradley Chen, Michael D. Smith, "An Analysis of Dynamic Branch Prediction Schemes on System Workloads", *Proc. 23rd Annual ISCA*, 1996

[8] Intel, *Intel XScale Microarchitecture*, 2001

[9] C. Perleberg and A. Smith, "Branch Target Buffer Design and Optimization", *IEEE Transactions on Computers*, 42(4): pages 396-412, 1993.

[10] T.Juan, S.Sanjeevan and J.J.Navarro, "Dynamic History-Length Fitting: A third level of adaptivity for branch prediction", *Proceedings of the 25th Annual ISCA*, pp 155-166, June 1998.

[11] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. "Mibench: A free, commercially representative embedded benchmark suite" *IEEE 4th Annual Workshop on Workload Characterization*, pages 83–94, 2001.

[12] S. McFarling. "Combining branch predictors". *DEC WRL TN-36*, June 1993.

[13] C.-C. Lee, I.-C. Chen, and T. Mudge. "The bi-mode branch predictor". *Proceedings of MICRO-30*, Dec 1997.

[14] S. T. Pan, K. So, and J. T. Rahmeh. "Improving the accuracy of dynamic branch prediction using branch correlation" *Proceedings of ASPLOS V*, pages 76–84, Boston, MA, October 1992.

[15] P. Michaud, A. Seznec, and R. Uhlig, "Trading conflict and capacity aliasing in conditional branch predictors," *Proceedings of the 24th Annual ISCA*, pp. 292--303, 1997

[16] P. Chang, E. Hao, T. Yeh, and Y. Patt. "Branch classification: a new mechanism for improving branch predictor performance". *MICRO-27*, November 1994.

[17] J. Smith. "A study of branch prediction strategies" *Proceedings of the 8th Annual ISCA*, May 1981.

[18] Johny Lee and Alan Smith. "Branch prediction strategies and branch target buffer design" *Computer, 17(1):6-22*, 1984.

[19] D. Burger and T. M. Austin. "The SimpleScalar tool set, version 2.0." Technical Report 1342, *Computer Sciences Department*, University of Wisconsin-Madison, June 1997