Using Hybrid Branch Predictors to Improve Branch Prediction Accuracy in the Presence of Context Switches

Marius Evers Po-Yung Chang Yale N. Patt

Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122
email: {olaf,pychang,patt}@eecs.umich.edu

Abstract

Pipeline stalls due to conditional branches represent one of the most significant impediments to realizing the performance potential of deeply pipelined, superscalar processors. Many branch predictors have been proposed to help alleviate this problem, including the Two-Level Adaptive Branch Predictor, and more recently, two-component hybrid branch predictors.

In a less idealized environment, such as a time-shared system, code of interest involves context switches. Context switches, even at fairly large intervals, can seriously degrade the performance of many of the most accurate branch prediction schemes. In this paper, we introduce a new hybrid branch predictor and show that it is more accurate (for a given cost) than any previously published scheme, especially if the branch histories are periodically flushed due to the presence of context switches.

Keywords: *branch prediction, context switch, superscalar, speculative execution*

1 Introduction

Branch prediction accuracy is a major performance factor in superscalar processor design. To improve branch prediction, various branch prediction strategies have been studied [13, 14]. These sophisticated branch predictors use branch history to achieve higher performance.

Recently, several hybrid branch predictors have been proposed that combine multiple prediction strategies into a single predictor [7, 2, 1]. These predictors use a selection mechanism to determine the most suitable component predictor for predicting each branch. Hybrid branch predictors have achieved higher prediction accuracies than single-scheme predictors by exploiting the strengths of each of their component predictors.

For hybrid branch predictors to achieve high prediction accuracy, they must be able to apply the most appropriate component branch predictor to each branch. Effective dynamic selection mechanisms have been proposed for hybrid branch predictors that consist of only two components [7]. For predictors with larger numbers of predictor components, a static selection mechanism, Branch Classification, has been proposed [2]. However, the effectiveness of this mechanism is limited because of its inability to adapt to changes during program execution.

Previous studies [9, 11] have shown that the performance of single-scheme predictors deteriorates when the branch history information is periodically destroyed. During the execution of programs, context switches may occur for various reasons, including I/O, page faults, end of time quantum, etc. Context switches can destroy the branch histories associated with particular processes; thus, the performance of branch predictors can significantly deteriorate.

This paper proposes a new dynamic selection mechanism for hybrid branch predictors with more than two single-scheme predictors. Increasing the number of different prediction schemes incorporated into the hybrid branch predictor increases the number of branches that it can accurately predict. In addition, the new selection mechanism allows us to also include predictors with shorter training times to assist the otherwise more accurate predictors during their warm-up phases. This allows the hybrid branch predictor to maintain a high prediction accuracy even after a loss of branch histories due to context switches.

This paper shows that the resulting hybrid predictor, the Multi-Hybrid, outperforms previously reported predictors.

ISCA '96 5/96 PA, USA
© 1996 ACM 0-89791-786-3/96/0005...$3.50

This paper is divided into five sections. Section 2 describes previously proposed branch prediction schemes. Section 3 describes the Multi-Hybrid predictor. Section 4 presents simulation results, comparing the Multi-Hybrid to other hybrid branch prediction schemes. Section 5 provides some concluding remarks.

2 Previous Work

To improve prediction accuracy, various branch prediction strategies have been studied. These prediction schemes can be divided into two groups: static and dynamic predictors.

Static branch prediction schemes use information gathered before program execution, such as branch opcodes or profiles, to predict branch direction. The simplest of these predicts that all conditional branches are always taken as in Stanford MIPS-X [4], or always not-taken as in Motorola MC88000 [8]. With additional hint bits in the branch opcodes, some processors [10] allow the compiler to pass prediction information to the hardware. The profile guided branch predictor bases its prediction on the direction the branch most frequently takes, which is determined by profiling the program on a training input data set [5].

Dynamic branch prediction algorithms use information gathered at run-time to predict branch direction. Smith [13] proposed a branch prediction scheme which uses a table of 2-bit saturating up-down counters to keep track of the direction a branch is more likely to take. Each branch is mapped via its address to a counter. The branch is predicted taken if the the most significant bit of the associated counter is set; otherwise, it is predicted not-taken. These counters are updated based on the branch outcomes. When a branch is taken, the 2-bit value of the associated counter is incremented by one; otherwise, the value is decremented by one.

By keeping more history information, a higher level of branch prediction accuracy can be attained [14]. Yeh and Patt proposed the Two-Level Branch Predictor which uses two levels of history to make branch predictions; the first-level history records the outcomes of the most recently executed branches and the second-level history keeps track of the more likely direction of a branch when a particular pattern is encountered in the first level history. The Two-Level Branch Predictor uses one or more k-bit shift registers, called branch history registers, to record the branch outcomes of the most recent k branches. It uses one or more arrays of 2-bit saturating up-down counters, called Pattern History Tables, to keep track of the more-likely direction for branches. The lower bits of the branch address are used to select the appropriate Pattern History Table(PHT) and the contents of the branch history register select the appropriate 2-bit counter to use within that PHT.

Several variations of the Two-Level Branch Predictor have been proposed [15]. McFarling [7] introduced gshare, a variation of the global-history Two-Level Branch Predictor which XORs the global branch history with the branch address to index into the PHT. Since the same global history patterns can occur for different branches during program execution, the global history pattern can be less efficient at identifying the current branch than the branch address itself. The gshare scheme tries to better identify the machine execution states by using both the branch address and the branch history. Lee and Smith [6] proposed a scheme where the value of each Pattern History Table entry is determined statically, using profile information; this scheme is referred to as the PSg scheme by Yeh and Patt [16]. Sechrest et al. [12] introduced another method, PSg (algo), of statically determining the values in the PHT and showed that a statically determining table could perform nearly as well as an adaptive PHT for short branch histories. Since the contents of PHT are determined statically, the PSg scheme trades the benefits of having the ability to adapt for the benefits of having no PHT warm-up time and simpler implementation.

To further improve prediction accuracy, hybrid branch predictors have recently been proposed [7, 2, 1, 3]. A hybrid branch predictor is composed of two or more single-scheme predictors and a mechanism to select among these predictors. A hybrid branch predictor can exploit the different strengths of its single-scheme component predictors, enabling it to achieve a prediction accuracy greater than that achieved by any of its components alone. McFarling [7] proposed a selection mechanism that combines two branch predictors using an array of 2-bit saturating up-down counters to keep track of which predictor is currently more accurate for each branch; each branch is mapped to a counter via its address. The counter is incremented based on the rule shown in Table 1. The most significant bit of the counter determines which one of the two predictors to use.

Predictor 1	Predictor 2	Update to Counter
Correct	Correct	No Change
Correct	Incorrect	Increment
Incorrect	Correct	Decrement
Incorrect	Incorrect	No Change

Table 1: Counter Update Rules

Chang et al. [2] proposed branch classification as another method to construct hybrid branch predictors. Branch classification allows an individual branch instruction to be associated with the branch predictor best suited to predict its direction. Using this approach, a hybrid branch predictor can be constructed such that each component branch predictor predicts

those branches for which it is best suited. By classifying branches based on their dynamic taken-rates, they proposed a hybrid branch predictor which uses the profile-guided predictor for branches that are mostly-taken or mostly-not-taken and McFarling's hybrid predictor for the remaining branches.

Chang and Banerjee[1] proposed the AVG predictor which can accurately predict loop branches. The AVG predictor keeps track of the average number of iterations executed for each loop. A branch is then predicted to exit the loop on the ith occurrence of that branch, where i is the average number of iterations associated with this loop. With branch classification, this predictor can be used to handle loop branches.

In this study, our hybrid branch predictor attempts to combine the advantages of previously described single-scheme predictors. We compare its performance with previously known hybrid branch predictors.

3 Multiple Component Hybrid Branch Predictor

Researchers have shown that the most effective single-scheme predictors use a large amount of branch history information and that two such predictors combined can outperform a single predictor at the same implementation cost. Furthermore, if you take into account context switches, some branch predictors that keep a large amount of history will periodically go through a warm-up phase in which they do not predict very accurately. In this warm-up phase, predictors with a smaller amount of history may be more accurate. The benefit of having multiple large predictors as well as small predictors argues for the need to include more than two component predictors in a hybrid branch prediction scheme. Thus, we propose using a multiple component hybrid predictor(Multi-Hybrid).

3.1 Selection Mechanism

Previous dynamic selection mechanisms are limited to selecting between at most two single-scheme predictors. To implement the selection mechanism for a N component Multi-Hybrid predictor, we propose adding N 2-bit up-down counters to each entry in the branch target buffer(BTB). These counters, the Predictor Selection Counters, keep track of the most accurate component predictor for each branch. Figure 1 shows the structure of the predictor selection mechanism using 2-bit up-down counters. Each of these counters is associated with a particular single-scheme branch predictor. They are updated in the following way:

- For a new entry in the BTB, all the counters are initialized to 3.
- All the predictors will generate a prediction. The prediction from the predictor which has the value 3 in its corresponding Predictor Selection Counter is used. When more than 2 selection counters contain the value 3, a priority encoder is used to decide which of the predictions to use.
- The counters get updated when the branch is resolved; if one of the predictors that had the value 3 in its selection counter was correct, the selection counters for all the incorrect predictors are decremented. Otherwise, the selection counters for all the correct predictors are incremented.

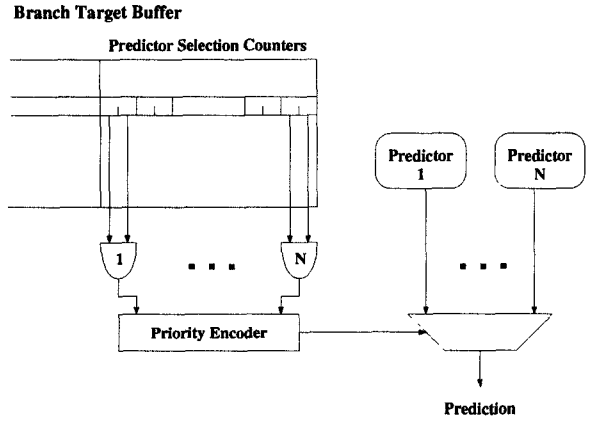

Figure 1: Predictor Selection Mechanism.

Using this updating strategy guarantees that at least one of the Predictor Selection Counters will be 3, simplifying our predictor selection mechanism. Our predictor selection mechanism also captures more information than saturating counters because it can better differentiate which of the component predictors are currently more accurate for each branch. For example, given the same initial counter values, the selection mechanism can differentiate a predictor that has been correct for the last 5 times from a predictor that has been correct for the last 4 times, while saturating counters can not.

If the selection mechanism is on the critical path, a variation can be implemented to reduce the time required to make a prediction. The priority encoding can be computed before the branch is fetched and stored in the BTB. Thus, when the branch is fetched, the previously calculated priority encoding is used to select the appropriate prediction; the resulting selection mechanism will require only one extra mux delay for choosing the appropriate prediction from the component predictors.

3.2 Component Predictors

Chang et al [3] showed that the gshare and PAs combination effectively exploits both inter-branch and intra-branch correlation. The gshare component is able to accurately predict branches whose outcomes are dependent on the outcomes of other static branches. The PAs component is able to accurately predict branches

whose outcomes are dependent on previous outcomes of the same static branch. Because of their different strengths, variations of both the global and per-address Two-Level Branch Predictors are used as components in the Multi-Hybrid. Instead of using PAs as our per-address predictor, we use pshare, a variation of the PAs scheme. This is because our initial results indicated that pshare is more suitable for the Multi-Hybrid.

However, one disadvantage of using large Two-Level Branch Predictors with long branch history registers is the long predictor warm-up time. Since context switches periodically destroy the histories maintained by such predictors, their performance can deteriorate significantly if context switches occur frequently.

To tolerate the effect of context switches, we propose the addition of static predictors and small dynamic predictors to the Multi-Hybrid. Since static predictors do not require any history information, their prediction accuracy is independent of context switches. This enables them to achieve a higher prediction accuracy than the large predictors shortly after the occurrence of context switches. As the number of branches executed after a context switch increases, the large dynamic predictors will become more accurate. The small dynamic predictors are included to provide a smooth transition from the static schemes to the large dynamic branch predictors.

Chang and Banerjee [1] showed that the AVG predictor is particularly effective in capturing the behavior of regular loop branches. The Multi-Hybrid includes the AVG predictor because of its superior accuracy in predicting these branches.

4 Experiments

We simulated the Multi-Hybrid and other relevant predictors for various predictor sizes and context switch intervals. We will first describe our simulation methodology and predictor model. We will then present our experimental results in two parts: the performance of alternative implementations of the Multi-Hybrid, and the performance of the Multi-Hybrid versus that of existing branch predictors.

4.1 Simulation Methodology

The performance of each hybrid predictor configuration was measured by trace-driven simulations, using a Motorola MC88110 instruction level simulator. The results presented in this paper are for the six SPECint92 benchmarks. Each benchmark was simulated until completion. For each benchmark, Table 2 lists the reference data set that was used and the exact number of instructions and conditional branches simulated.

[1] Abbreviated version of the SPECint reference input set int_pri_3.eqn. It consists of 15 boolean equations with 39 different variables.

Benchmark	Input	# of Instrs	# of Cond Branches
espresso	bca.in	354645678	74622690
xlisp	7 queens	218806680	32765434
eqntott	int1.eqn[1]	192782895	26099824
compress	in	86445440	11178108
sc	loada1	143176494	30484852
gcc	stmt.i	107163324	17252822

Table 2: Summary of the SPECint92 Benchmarks along with the input data sets.

4.2 Predictor Model

Our branch predictor model assumes that during context switches, all history information associated with the hybrid branch predictor is lost. To model this, the counters for the predictor selection mechanism, the branch history table, and the pattern history tables are all reinitialized. We believe flushing the branch history information is a reasonable model of context switches on a moderately to a heavily loaded system.

For our selection mechanism, if a branch misses in the BTB, the gshare predictor is used to predict this branch.

4.3 Predictor Configurations

For a given hardware cost, the hybrid predictor configuration is specified by the single-scheme predictors used, the amount of hardware devoted to each scheme, and the predictor selection mechanism. In this section, we will examine the performance impact of each component predictor, the priority ordering scheme, and the selection counters.

4.3.1 Component Predictor Selection

The most accurate single-scheme predictors use a large amount of history information. Predictors with shorter warm-up times, such as static predictors and small dynamic predictors, can achieve a higher prediction accuracy than the large predictors immediately after the occurrence of a context switch. In our Multi-Hybrid, we therefore allocate resources for implementing both classes of predictors.

Since the most accurate predictors use a larger amount of hardware, we allocate approximately 2/3 to 3/4 of the hardware budget for these predictors. Per-address and global variations of the Two-Level Branch Predictor have been shown to be an effective combination; therefore, we allocate approximately half of our total budget for gshare, the most accurate single-scheme predictor, and a quarter of the total budget for pshare.

The static predictors, always taken and always not-taken, and the small dynamic schemes, 2bC and PSg,

are included because of their small implementation costs and their ability to more accurately predict branches immediately after flushing of the branch histories.

When the gshare scheme is of a moderate or large implementation cost requiring a long warm-up time, a smaller global history scheme is included to provide a smooth transition between the small and the large predictors. We use GAs for this smaller predictor.

Although the loop predictor addresses a separate class of branches, it is only included for the three larger configurations due to its implementation cost.

The initial configurations for the Multi-Hybrid are determined using the above model. To improve the cost/performance of the Multi-Hybrid, we examined the contribution of each component predictor to the overall performance of the Multi-Hybrid. We measured and compared the performance of the Multi-Hybrid with and without each component predictor. The results of these experiments showed that the component predictors described above, with the exception of PSg and Always Not-taken, contribute to the performance of the Multi-Hybrid for one or more of the benchmarks. Because the PSg and Always Not-taken predictors only contribute marginally, they are not included in the Multi-Hybrid for the following experiments.

Table 3 shows the resulting Multi-Hybrid configurations. Each column shows the approximate hybrid predictor size and the sizes of each of its components. The cost models for, and short explanations of, the single-scheme predictors are given in Table 4. Since a 2K entry BTB is used in this study, the cost of the selection mechanism is estimated to be $2^{11} \times c \times 2$ bits, where c is the number of component predictors in the Multi-Hybrid. For hybrid predictors with implementation cost 18 KBytes or less, some of the smaller predictors had to be excluded to fit our hardware budget, as indicated in Table 3 with a '-'.

Since we have not exhaustively studied all predictor configurations, our configurations are not guaranteed to be optimal. However, our choices are sufficiently constrained to allow significant investigation of the Multi-Hybrid. Future work will expand on these choices.

4.3.2 Priority Ordering

The performance of the selection mechanism may depend on the priority ordering of the component predictors of the Multi-Hybrid. To determine the impact of the priority ordering of the branch predictors on the performance of the Multi-Hybrid, we simulated the 6! possible priority orderings for those predictors with six components and the 5! possible priority orderings for those predictors with five components. We measured the prediction accuracy over all 6 benchmarks for three context switch intervals (16000, 256000 and no context switches). Table 5 shows the performance of both the optimal priority ordering and the priority ordering used in our experiments. We considered the optimal priority ordering to be that which had the highest average prediction accuracy over the 18 runs. Our priority ordering is shown in Table 3 where the component predictors are listed based on their priority, starting with the predictor of the highest priority. That is, the 2bC scheme has the highest priority and the Always taken scheme has the lowest priority. Our priority ordering yields a negligible decrease in performance compared to the optimal ordering.

	Hybrid Predictor Size (KBytes)				
	~11	~18	~33	~64	~116
	Cost of each component				
Selection Mechanism	2	2.5	3	3	3
2bC	.5	.5	.5	.5	.5
GAs	-	2	2	4	8
gshare	4	8	16	32	64
pshare	4	5.25	7.5	20	36.25
loop	-	-	4	4	4
Always taken	0	0	0	0	0

Table 3: Multi-Hybrid Configurations

Predictor Size	Actual	Optimal
~18 KB	95.22	95.22
~64 KB	95.65	95.65

Table 5: Impact of Priority Ordering

4.3.3 Predictor Selection Counters

The performance of the selection mechanism also depends on the type of Predictor Selection Counters used to keep track of the most accurate component predictor for each branch. In addition to using 2-bit up-down counters in the selection mechanism as discussed in section 3.1, we also measured the performance of the Multi-Hybrid when using 3-bit up-down counters. Table 6 compares the performance of 2-bit counters with that of the 3-bit counters for a Multi-Hybrid of size 18 KByte when no context switches are modeled. With the exception of compress, 2-bit up-down counters perform comparably to 3-bit up-down counters in capturing the most accurate predictor for each branch. The results for a Multi-Hybrid of size 64 KByte also show that the performance for 2-bit and 3-bit up-down counters is similar. For the following experiments, we only consider the Multi-Hybrid using 2-bit up-down counters.

Predictor	Algorithm	Cost (bits)
2bC	the two bit counter predictor [13] consisting of an 2K entry array of two bit counters.	2^{12}
GAs(m,n)	the global variation of the Two-Level Adaptive Branch Predictor [15] consisting of a single m-bit global branch history and n pattern history tables.	$m + 2^{m+1}n$
PSg(m)	a modified version of the per-address variation of the Two-Level Adaptive Branch Predictor [16, 12] consisting of 2K m-bit branch history registers and a single pattern history table (each PHT entry uses one statically determined hint bit instead of a 2bC). The version of PSg used in this study is the PSg(algo) [12].	$2^{11}m + 2^m$
gshare(m)	a modified version of the global variation of the Two-Level Adaptive Branch Predictor [7] consisting of a single m-bit global branch history and a single pattern history table.	$m + 2^{m+1}$
pshare(m)	a modified version of the per-address variation of the Two-Level Adaptive Branch Predictor [15] consisting of 2K m-bit branch history registers and one pattern history table. As in the gshare scheme, the branch history is XORed with the branch address to select the appropriate PHT entry.	$2^{11}m + 2^{m+1}$
loop(m)	an AVG predictor[1] where the prediction of a loop's exit is based on the iteration count of the previous run of this loop. A 2K entry array of two m-bit counters is used to keep the iteration counts of loops. In this study, $m = 8$.	$2^{12}m$
Always taken		0
Always not-taken		0

Table 4: Hardware costs for the single-scheme predictors.

Predictor Size	Benchmark	2-bit	3-bit
~18KB	compress	92.98	93.18
	eqntott	98.42	98.42
	espresso	97.00	97.03
	gcc	94.30	94.30
	sc	98.13	98.14
	xlisp	98.15	98.21

Table 6: Predictor Selection Counters

4.4 Performance of the Multi-Hybrid

In this section, we compare the performance of the Multi-Hybrid to that of the PAs/gshare and 2bC/gshare schemes. The PAs/gshare and the 2bC/gshare schemes are hybrid branch predictors that combine gshare with PAs and gshare with 2bC (the 2-bit counter scheme) respectively [7]. For these two schemes, 2048 2-bit counters are used for selecting the more accurate component predictor at run-time.

We first analyze performance for the branch predictors when no context switches are modeled. We then show how periodically flushing the branch histories due to context switches affects the branch prediction accuracy.

4.4.1 Prediction Accuracy Without Context Switches

Figure 2 compares the prediction accuracies of the three hybrid branch predictors at various implementation costs. For illustration purposes, the figure also includes the prediction accuracy of the best single-scheme predictor, gshare. The Multi-Hybrid proposed in this paper outperforms the other hybrid predictors for all predictor sizes that were examined. For an implementation cost of approximately 64 KBytes, the Multi-Hybrid achieves a 97.13% prediction accuracy, as compared to 96.63% for the PAs/gshare scheme.

Figure 2 also shows that the performance of the 33 KByte Multi-Hybrid predictor is significantly better than that of its 18 KByte counterpart. For predictors with sizes of 18 KBytes or smaller, the loop predictor was not included as part of the Multi-Hybrid predictor because of the smaller hardware budget. For predictors with implementation costs of 33 KBytes or larger, the addition of the loop predictor resulted in a large

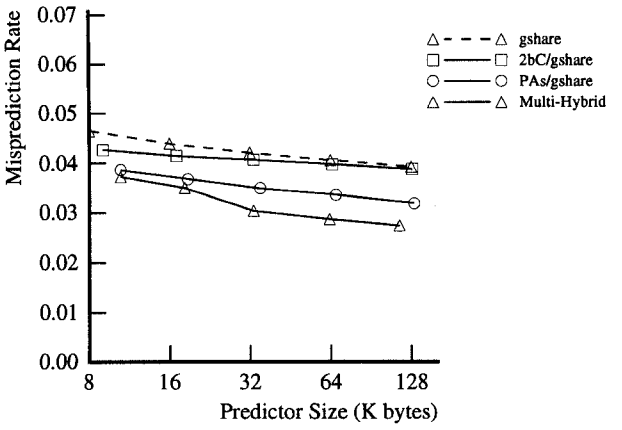

Figure 2: Prediction Accuracy averaged over all benchmarks when Context Switches are not Modeled.

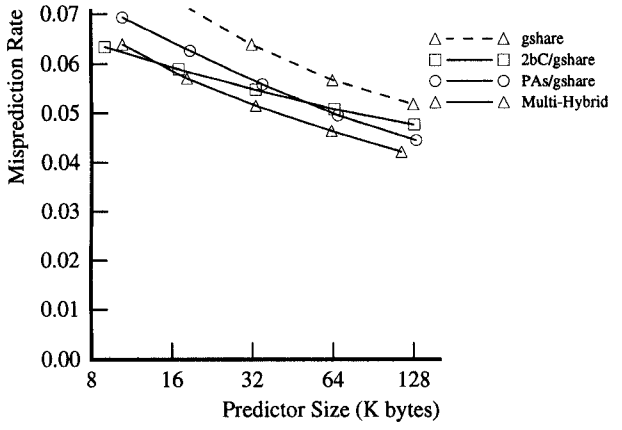

Figure 3: Prediction Accuracy of gcc when Context Switches are not Modeled.

increase in performance, in particular for the eqntott benchmark.

Our results for the individual benchmarks show similar trends.

Figure 3 shows the prediction accuracies of these branch predictors for the gcc benchmark. At an implementation cost of approximately 64 KBytes, the Multi-Hybrid outperforms the PAs/gshare scheme, improving prediction accuracy from 95.04% to 95.36%.

4.4.2 The Effect of Context Switches on Prediction Accuracy

Since context switches periodically destroy the history maintained by predictors, the performance of history-based predictors can deteriorate significantly if context switches occur frequently. Figures 4 and 5 show how context switches as modeled by periodic flushings affect hybrid predictors with implementation costs of approximately 18 and 64 KBytes respectively. The solid

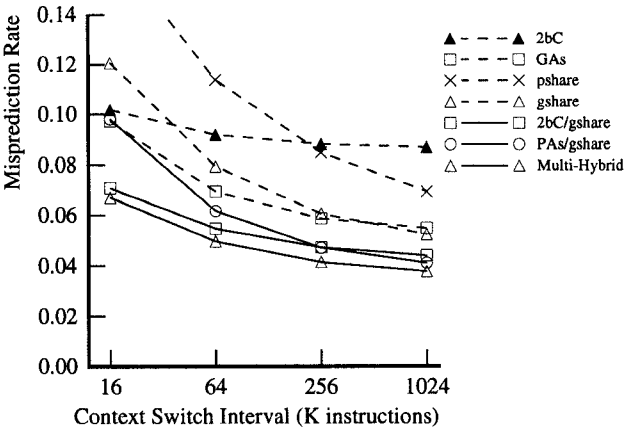

Figure 4: The Effect of Periodic Flushing on Hybrid Predictors with Implementation Cost 17 to 19 KBytes averaged aver all benchmarks.

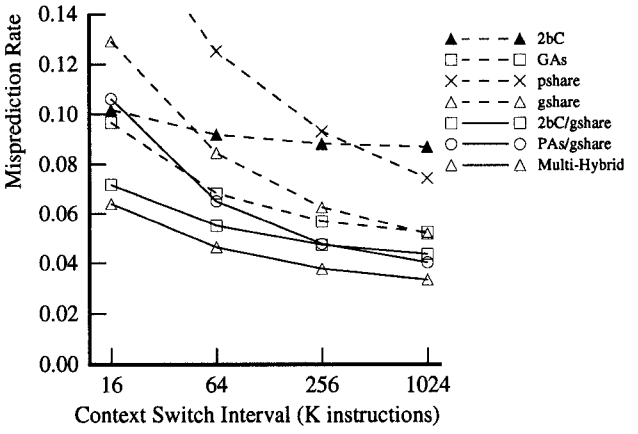

Figure 5: The Effect of Periodic Flushing on Hybrid Predictors with Implementation Cost 64 to 67 KBytes averaged aver all benchmarks.

lines show the performance of hybrid branch predictors as a function of context switch interval.

These graphs also show the performance of the single-scheme predictors that are used as component predictors in the Multi-Hybrid. Their performance as a function of context switch interval is shown by the dotted lines. The size of each component predictor is shown in Table 4 of Section 4.3. The loop predictor is not shown in Figures 4 and 5 because it was not designed to be used as a single-scheme predictor by itself but as a component in a hybrid predictor with other single-scheme predictors.

As shown in Figures 4 and 5, the Multi-Hybrid outperforms the other prediction schemes for all context switch intervals examined. With context switch intervals of 256,000 instructions and a predictor size of approximately 64 KBytes, the Multi-Hybrid predictor achieves a prediction accuracy of 96.22% whereas

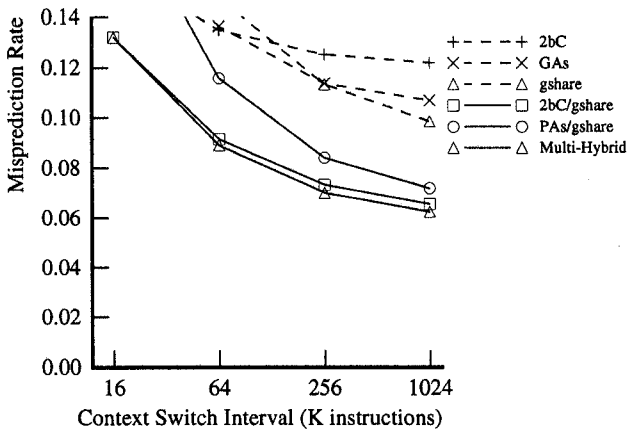

Figure 6: The Effect of Periodic Flushing on Hybrid Predictors with Implementation Cost 17 to 19 KBytes. (gcc)

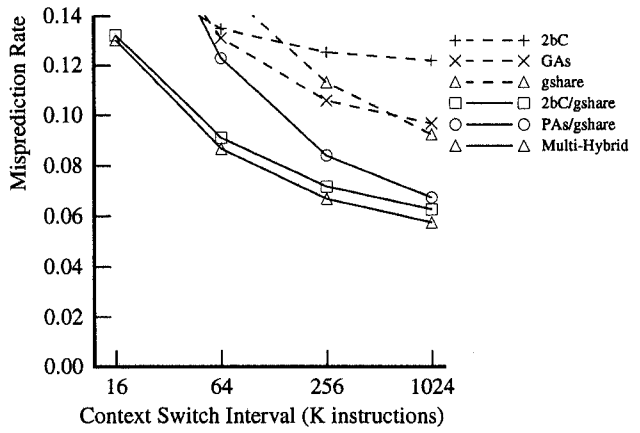

Figure 7: The Effect of Periodic Flushing on Hybrid Predictors with Implementation Cost 64 to 67 KBytes. (gcc)

PAs/gshare and 2bC/gshare achieve 95.22% and 95.26% respectively, reducing the number of mispredictions by over 20%. For smaller predictors with implementation costs around 18 KBytes, the corresponding reduction in mispredictions is approximately 12%.

Figures 6 and 7 show how context switches affect hybrid predictors with implementation costs of approximately 18 and 64 KBytes respectively on the gcc benchmark. With context switch intervals of 256,000 instructions and a predictor size of approximately 64 KBytes, the Multi-Hybrid predictor achieves a prediction accuracy of 93.31% whereas PAs/gshare and 2bC/gshare achieve 91.6% and 92.82% respectively, reducing the number of mispredictions by 20.36% and 6.82%. For smaller predictors with implementation costs around 18 KBytes, the corresponding reduction in mispredictions are approximately 16.67% and 4.38%.

Figures 4 through 7 also show that the performance of the single-scheme 2bC is better than that of the Two-Level Branch Predictors for context switch intervals of 16,000 instructions. In addition, the performance of 2bC is less sensitive to the length of the context switch intervals. Extrapolating from these graphs, the performance advantage of 2bC over the Two-Level Branch Predictors increases as the length of the context switch intervals decreases; for very short context switch intervals when the predictors have little or no time time to warm up, the 2bC scheme can outperform the Two-Level Branch Predictors. Since the 2bC/gshare scheme utilizes the 2bC scheme, it was able to outperform the PAs/gshare scheme when context switch intervals are below 256,000 instructions.

On the other hand, the PAs/gshare scheme outperforms the 2bC/gshare scheme when the context switch intervals are above 256,000 instructions. Because the PAs/gshare scheme uses both the PAs and the gshare schemes, it is more effective than 2bC/gshare in exploiting both the inter-branch and intra-branch correlation.

Finally, our Multi-Hybrid branch predictor is able to combine the advantages of all three single-scheme predictors, using 2bC to tolerate context switch and gshare with pshare to achieve higher prediction accuracy. In addition, the Multi-Hybrid can also include other single-scheme predictors, such as the loop predictor, allowing it to better handle special classes of branches.

5 Conclusions

In this paper, we have introduced a new hybrid branch predictor, the Multi-Hybrid. By combining branch predictors with different characteristics, we produced a hybrid predictor that predicts a wider range of branches correctly while being less sensitive to periodic flushing of the history information.

When context switches were modeled, the Multi-Hybrid predictor significantly outperformed the PAs/gshare and the 2bC/gshare schemes. At a context switch rate of 1 per every 256,000 instructions, the Multi-Hybrid shows a reduction of 12.5% and 20.6% in mispredictions for 18 KByte and 64 KByte predictor sizes respectively on the SPECint92 benchmarks.

When context switches were not modeled, the Multi-Hybrid predictor also achieved an average reduction of 9% in branch mispredictions over the PAs/gshare scheme and 20% over the 2bC/gshare scheme for 11KB to 64KB predictor sizes on the SPEC benchmarks.

We also showed that the PAs/gshare scheme that was closest to the Multi-Hybrid when not accounting for context switches is very vulnerable to periodic flushing of the history information due to context switches, even when those context switches happen at large intervals.

6 Acknowledgments

We gratefully acknowledge the support of our industrial partners, without which it would not have been possible to undertake this work, in particular Intel Corporation and NCR Corporation.

References

[1] P. Chang and U. Banerjee, "Profile-guided Multiheuristic Branch Prediction", *Proceedings of the International Conference on Parallel Processing*, July, 1995.

[2] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y.N. Patt, "Branch Classification: a New Mechanism for Improving Branch Predictor Performance", *27th ACM/IEEE International Symposium on Microarchitecture*, Nov. 1994.

[3] P.-Y. Chang, E. Hao, and Y.N. Patt, "Alternative Implementations of Hybrid Branch Predictors", *28th ACM/IEEE International Symposium on Microarchitecture*, Nov. 1995.

[4] P. Chow and M. Horowitz, "Architecture tradeoffs in the design of MIPS-X," *Proceedings of the 14th Annual International Symposium on Computer Architecture*, June 1987.

[5] J. A. Fisher and S. M. Freudenberger, "Predicting Conditional Branch Directions from Previous Runs of a Program", *5th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1992.

[6] J.K.F. Lee and A.J. Smith, "Branch Prediction StrategiesBranch Target Buffer Design," *IEEE Computer*, pp.6-22, January 1984.

[7] S. McFarling, "Combining Branch Predictors", WRL Technical Note TN-36, Digital Equipment Corporation, June 1993.

[8] C. Melear, "The design of the 88000 RISC family," *IEEE MICRO*, pp.26-38, April 1989.

[9] R. Nair, "Dynamic Path-Based Branch Correlation", *28th ACM/IEEE International Symposium on Microarchitecture*, Nov. 1995.

[10] The PowerPC Architecture: A Specification for a New Family of RISC Processors, Ed. C. May et al, Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1994.

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

28th ACM/IEEE International Symposium on Microarchitecture, Nov. 1995.

[12] S. Sechrest, C.-C. Lee, and Trevor Mudge, "The Role of Adaptivity in Two-Level Adaptive Branch Prediction," *28th ACM/IEEE International Symposium on Microarchitecture*, Nov. 1995.

[13] J.E. Smith, "A Study of Branch Prediction Strategies," *8th International Symposium on Computer Architecture*, June 1981.

[14] T.-Y. Yeh and Y.N. Patt, "Two-level Adaptive Branch Prediction," *24th ACM/IEEE International Symposium on Microarchitecture*, Nov. 1991.

[15] T.-Y. Yeh and Y.N. Patt, "Alternative Implementations of Two-level Adaptive Branch Prediction," *19th Annual International Symposium on Computer Architecture*, May 1992.

[16] T.-Y. Yeh and Y.N. Patt, "A Comparison of Dynamic Branch Predictors that Use Two Levels of Branch History", *20th Annual International Symposium on Computer Architecture*, May 1993.