to appear in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT) 1998

Static Methods in Hybrid Branch Prediction

Dirk Grunwald, Donald Lindsay, and Benjamin Zorn

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309–0430 USA

Abstract

Hybrid branch predictors combine the predictions of multiple single-level or two-level branch predictors. The prediction-combining hardware — the "meta-predictor" — may itself be large, complex and slow. We show that the combination function is better performed statically, using prediction hints in the branch instructions. The hints are set by profiling or static analysis. Although the meta-predictor is static, the actual predictions remain dynamic, so there is little risk of worst-case performance. An important advantage of our approach is that a branch site only causes interference within a single component predictor, reducing capacity demands. We argue that our proposal is implementable, and that it addresses the scaling issues currently facing hardware designers.

We show that the static hybrid method we propose is more effective than existing techniques based on dynamic selection, and requires less hardware. For example, one result shows a conventional 4096-bit dynamic selection mechanism getting a 4.7% average miss rate, while our static approach gets 3.6%. These results are obtained with the Instruction Benchmark Suite (IBS), a realistic whole-system benchmark, and the SPECint95 suite, using realistic hardware sizes. All the results we present are based on a cross-validation methodology, in which the profile data used for static selection are based on training inputs that are entirely different from the inputs used to evaluate the performance of the technique.

1 Introduction

Branch prediction is considered to be an important research topic because misprediction penalties have become a major impediment to high processor utilization. Branch prediction is used to allow speculative execution in pipelined processors. Pipeline stalls on a modern wide-issue processor introduce a large lost-opportunity cost, meaning that several instructions were not issued, and branch recovery mechanisms are difficult to implement because they typically require checkpointing of considerable resources. Ranganathan and Jouppi [13] argue that branch prediction will be the most limiting aspect of computer architectures, surpassing the limitations of memory systems by 2010. Their observations were true even when they assumed that 75% of all predictions were predicted using an "oracle" predictor, with only the remaining 25% being derived from an actual predictor (a McFarling branch predictor with a 32K-entry PAs predictor, a 32K-entry gshare predictor and a 32K-entry meta-predictor).

Recently there has been interest in hybrid branch predictors. The fundamental idea here is that different predictor schemes have different advantages. Hence, a combination of basic schemes might (at some cost) obtain all of their advantages. The upcoming Alpha 21264 [6, 9] uses a large hybrid predictor.

This paper proposes that the prediction-combining hardware, or selection mechanism, be replaced by a static choice encoded in the branch instruction. We feel this paper makes the following contributions. We show that static hybrid methods out-perform dynamic-hybrid branch predictors, even when the static-hybrid branch predictor uses fewer hardware resources. We show *why* the static hybrid technique works well. Static hybrid methods increase capacity and decrease aliasing in branch predictors, since components can view subsets of the workload stream, but still have the benefit of dynamic branch predictors. Lastly, we show that our results are never worse than those of the dynamic selection mechanism based on a geometric mean over the IBS and SPECint95 benchmark suites, even when our profile information is sometimes of poor quality.

This paper is organized as follows. In Section 2, we survey related work and describe some of the more important terms and prediction schemes. In Section 3, we introduce our proposal, and explore hardware and software implementation issues. In Section 4, we describe sample hardware designs and the benchmarks we use to evaluate them. In Section 5 we evaluate the different predictors and demonstrate an improvement relative to a comparable dynamic hybrid predictor. For example, one result, based on a geometric mean over the IBS benchmark, shows a 4096-bit dynamic selection mechanism getting a 4.7% average miss rate, while our static approach has a 3.6% miss rate. In Section 6 we summarize and discuss some possibilities for future work.

2 Previous Work

Because branch prediction is such an important part of modern architectures, there is an active and growing body of research in the area, to the point where a 1992 taxonomy [19] is already becoming dated. Branch predictors generally use one or more of the program counter, recent local branch history, and recent global branch history, to index into an array of up/down saturating counters. The actual prediction is based on whether the counter contains a high or low value. Counters typically use two bits to introduce hysteresis that improves prediction for loop-exit branches.

A branch history register (BHR) is a shift register representing recent branch outcomes where one bit is updated as each branch commits. The gshare predictor uses a single global branch history, but other techniques, such as PAs or SAg predictors [19] use a number of branch history registers organized as a *branch history table* (BHT), and subdivide the branch stream to these different BHR's. The PAs variant uses "tags" to associate the BHR with a specific branch, while the SAg variant does not use tags, making it less expensive to implement at the expense of introducing aliasing in the BHT. A common implementation of the PAs predictor, suggest by Yeh and used in the Intel PentiumPro, is to store the BHR in the branch target buffer (BTB), allowing the branch predictor to share the tags with the BTB. A BHT is usually organized as a 2-way or 4-way associative structure. Throughout this paper, we will use a PAs branch history table that uses tags.

McFarling [11] introduced the idea of hybrid predictors. He describes a series of simple predictors, most notably "gshare". In the gshare predictor the BHR and program counter are mixed using an exclusive-or to index the *pattern history table* (PHT), a table of 2-bit saturating counters. In the PAs predictor, the program counter is used to select a branch history register and that BHR is used to select the pattern history register. McFarling explains why each simple predictor works, and concludes that to some extent each works in a different way – the gshare predictor is good at capturing patterns across multiple branches, while the PAs predictor is good at capturing the behavior of a single branch. Thus, McFarling proposed a "combining predictor" that contains two *component predictors* and a *meta-predictor* that selects between the different component predictors. In McFarling's design, the meta-predictor was a table of 2-bit saturating counters indexed by the program counter. The counters reflect the states "strongly gshare", "weakly gshare", "weakly PAs" and "strongly PAs". When a branch is predicted, each component predictor is queried, and the state of the meta-predictor is used to select the component that is actually used for the branch prediction. When the branch actually commits, *both* component predictors are updated, and the meta-predictor is updated to favor the component predictor that had the correct prediction. McFarling concludes from his comparisons over the SPEC89 benchmark suite that combining predictors have better performance than any single predictor of the same hardware size, even when the size of the meta-predictor is included in the comparison.

In general, multiple component predictors can be used, and different designs can be used for the meta-predictor. Chang, Hao, Yeh and Patt proposed *branch classification* [2], which used a static meta-predictor and a static prediction component along with a dynamic component. The static meta-predictor is used to select either the dynamic or static component. The static predictor component was set using an edge-profile, and the prediction was set to the most likely branch outcome. The meta-predictor was set using the same edge profile. Branches that had a e.g., $\geq 95\%$ static prediction accuracy used the static predictor, and the other branches were assigned to the dynamic predictor. We compare our static-hybrid method to branch classification using one of the most successful hybrid predictors proposed, PG+gshare, which combines a static prediction and a gshare predictor. The selection threshold was set to a 95% prediction accuracy.

In other work, Chang and Banerjee [3] investigate the use of static hybrid methods in choosing between a new component predictor they propose (AVG) and another 2-level predictor. In later work, Chang, Hao and Patt [1] also search for good combinations of components. To reduce the design space, they used components of equal size and determined the effectiveness of each component for each branch site. This allowed a static computation of the miss rate of an "idealized" hybrid predictor. They argued that this ideal is not perfect since it does not adapt to program dynamics. However, the method allowed them to choose the best pairing, which they determined to usually be gshare plus PAs [14, 19]. Like McFarling, they argue that the pairing is successful because gshare does well on site correlation, and PAs does well on successive uses of one site. They also suggest that some predictors have a quicker warm-up time (i.e., a smaller training cost). In a more recent paper Evers, Chang and Patt [8] consider a larger selection of component predictors, and propose a hybrid predictor with six, rather than two, components. This "multi-hybrid" contains unequal sized elements, in the expectation that some will warm-up or train more quickly after context switches. It also contains a component predictor that is specialized for predicting loops. Due to the complexity of the design, the selection mechanism used may introduce timing constraints, although the authors suggest ways to ameliorate such effects.

Recently, a different approach to dynamic hybrids has been proposed by Michaud, Seznec and Uhlig [12]. Their "gskew" predictor uses three component predictors, but does not have a meta-predictor. Instead, selection is done by majority vote. The components use different hash functions to combine the branch history register and program counter address. The objective is to divide aliasing into three disjoint sets, so that the majority vote will effectively eliminate problems caused by aliasing within any one set. One of the interesting aspects of the gskew predictor is that not all component predictors are updated when a branch is resolved. The static hybrid we propose also has this characteristic, as we describe in the next section.

3 Static Hybrid Predictors

A hybrid branch predictor combines two or more *component predictors* into a single branch prediction mechanism using a *selection mechanism*. With the static hybrid predictor, we propose that the dynamic selection mechanism used in dynamic hybrid branch predictors, such as McFarling's, be replaced by a *static per-branch selection bit*, much as in Branch Classification [2]. Like branch classification, we investigate statically choosing between different component predictors. However, in our model, both components are fully dynamic, and not as susceptible to worst-case effects caused by poor training inputs. Furthermore, as results in Section 5 suggest, use of branch classification requires establishing a careful balance between the dynamic component and the threshold for the static component. While Chang and Bannerjee [3] mentioned the static selection of components in their paper, the focus of their analysis was on the advantage of their AVG branch predictor, and they did not indicate any advantage to the use of a static meta-predictor.

Our work more directly investigates the use of static hybrid methods and quantifies the value gained by such an approach. Moreover, we use a different profiling mechanism – rather than simply collect edge profiles, we use branch predictor simulations to select the appropriate component predictors. We simulate both component predictors on every branch with the assumption that each component predictor is fully updated. After the training phase is complete, we select the component predictor for each branch that was the most effective overall. This profile operation is somewhat more complex than standard basic block or instruction count profiles, since a simulation of all the alternative component predictors must be performed. However, this information could easily be gathered using on-line performance tools [5, 4] that provide stochastic instruction sampling, or, as done for this paper, using binary instrumentation [15].

Since the component predictors are selected statically, we only need to update the component predictor that is actually used. This reduces the interference from capacity misses or aliasing in the different components. Specifically, we always update the gshare global history register on every branch, but do not update the component predictor counter arrays unless the branch has been statically assigned to that component. Since the initial simulation updates *both* components, that initial assignment may influence the benefit of one predictor over the other, and thus affect the assignment of branches to specific predictors. When using binary instrumentation, we can use multiple iterations of the training process to refine the static assignment. We found this improved performance slightly, but was not essential. With online tools, such as ProfileMe [5], we would expect that continuous sampling and re-optimization would have a similar effect. All of our results are shown without this improvement.

Because our method allows components to be updated selectively, we outperform the McFarling predictor in almost all cases because the partial update reduces the load on each of the dynamic components, effectively increasing their capacity and thus performance. A similar effect was noted in the *bimode* predictor [10].

4 Experimental Design

In this section, we discuss the experimental design that we used to obtain the results in the following section. Specifically, we discuss the configuration of the static hybrid and McFarling component predictors evaluated, the two program benchmark suites, and the evaluation procedure, including the cross validation process.

Component Configurations In Section 2 we discussed several proposed dynamic hybrid predictors. From these, we chose to compare our static hybrid predictor with an instance of the McFarling dynamic hybrid predictor because the McFarling predictor is well known and published experimental results show that it works well [11, 1, 8] . Chang, Hao and Patt [1] reported that their best predictor combination was PAs [19] plus gshare [11]. Both of these predictors have several parameters, but papers such as Sechrest, Lee and Mudge [14] report the most effective ratios between the parameters.

Based on this prior work, we chose a PAs predictor with a 256-entry BHT, plus a one-dimensional array of 512 2-bit

IBS				*SPECINT95*				
Benchmark	*Millions of Instrs*	*% of Instrs in Kernel*	*Conditional Branch Sites*	*Benchmark*	*Millions of Instrs*		*Cond. Branch Sites*	
					train	*ref*	*train*	*ref*
groff	105	13.6	6292	compress	50	60156	383	396
gs	118	13.3	12761	gcc	1687	334	18145	16956
jpeg_play	151	8	7828	go	1	35971	271	5409
mpeg_play	99	21.8	5571	ijpeg	1964	41578	1285	1258
nroff	130	8.3	5230	li	216	66613	558	795
real_gcc	107	12.7	17300	m88ksim	138	82810	1088	1315
sdet	42	98.7	5284	perl	13	19934	1569	1574
verilog	47	22.6	4613	vortex	3325	95791	6381	6366
video_play	53	69.9	4575					

Figure 1. Characteristics of IBS and SPECint95 Suites

counters. Further, we chose a BHT with 4-way associativity, and an "update on taken" update policy. We chose a gshare predictor with 2^{11} counters, in a $2^7 \times 2^4$ organization, meaning that the 2-level predictor is organized to use 7 bits from the program counter and have a 4-bit BHR. Following McFarling's suggestions [11], the McFarling combining predictor has the same number of counters as the gshare component (2048 counters). The hardware configurations we have chosen for evaluation purposes are similar to current practice in production microprocessors. For instance, the MIPS R10000 [18] uses a bimodal [11] predictor of 512 counters. The Alpha 21164 [7] uses 2048 counters in the same organization: the Alpha 21264 [9, 6] contains four arrays, with 1024 or 4096 entries each.

Benchmark Suites To evaluate and compare the performance of our static hybrid predictor with that of the McFarling predictor, we chose two benchmark suites often used in branch prediction studies: the SPECint95 suite [16] and the the Ultrix portion of the IBS traces [17]. We chose to focus on the integer portion of SPEC95 in our evaluation because many of the branches in the SPECfp95 benchmarks are highly predictable and less interesting for the purpose of evaluating branch prediction hardware. In our evaluation, we present performance figures for the individual programs in each suite, as well as the geometric mean across each suite. Figure 1 contains a summary of some of the important characteristics of the two benchmark suites. We chose to use both suites because they each bring qualities that the other lacks.

Cross Validation Methods We used *cross validation* to validation our branch predictor, since it is partially based on information from program profiles. With cross validation, we used different input sets during the training and evaluation phases. For the SPECint95 applications, we used the "train" and "ref" inputs. The IBS traces do not provide multiple inputs so we split the traces in half, using one half ("A") for training and the other ("B") for evaluation. We then reversed the process, using "B" for training and "A" for evaluation. We then report the average of these two cross-validated results.

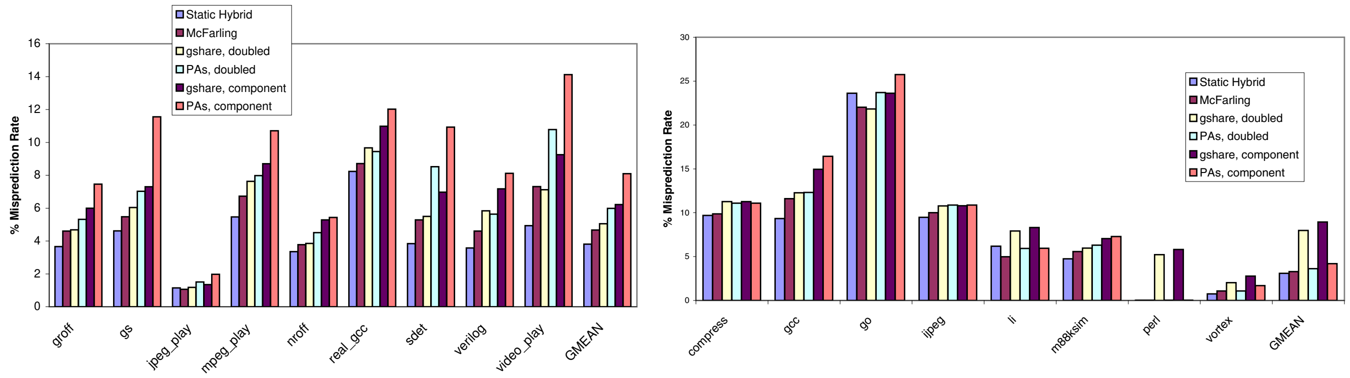
5 Results

The advantages of the static hybrid predictor are that it requires less hardware than a dynamic-hybrid predictor since component selection is done statically, instead of dynamically. Furthermore, since each branch is always predicted by the same component, only one of the components needs to be updated, effectively increasing the capacity of the hybrid predictor over one in which all components are updated. We feel the implementation is not likely to add any overhead over existing methods. However, there are potential disadvantages. The profiling mechanism is more complicated than for schemes such as branch classification, and unless an on-line mechanism such as ProfileMe is combined with continuous optimization, there is the potential that poor training can reduce the effectiveness of our approach. Finally, because component assignment is static, the approach does not adapt to programs with phases in which no one component assignment is optimal.

Overall Performance Comparison Figures 2(a) and 2(b) show a comparison of the performance of the proposed static hybrid branch predictor against a number of alternatives for the IBS and SPECint95 benchmarks, respectively. These alternatives include the McFarling dynamic hybrid predictor with exactly the same components as our static hybrid, as well as 2 configurations each of non-hybrid gshare and PAs predictors. Each of the PAs and gshare predictors are considered in sizes equal to the size of the individual components in the hybrid predictors, and also double that size, which approximates the total hardware size our static hybrid predictor (if the tag information present in the PAs component is not included). Specifically, the "gshare, doubled" bar shows the result of using a gshare predictor that is twice as large (e.g., with geometry $2^8 \times 2^4$ instead of $2^7 \times 2^4$). Similarly, "PAs, doubled" is just the PAs component with 512 BHT entries and the same number of counters.

We emphasize that the McFarling predictor has exactly the same predictor components, but also requires dynamic selection hardware, which is approximately the same size as the gshare component. Thus the McFarling predictor we are comparing against has a greater hardware cost than our static hybrid predictor.

The most important conclusion to reach from the figures is that the static hybrid predictor outperforms all the other predictors in all but three programs, "go," "li," and "jpeg_play". Furthermore, the geometric mean over both benchmark suites shows that the static hybrid has an advantage, especially in the case of the IBS suite.



(a) IBS

(b) SPECint95

Figure 2. The Static Hybrid predictor versus alternatives

The non-hybrid gshare and PAs predictors we compare against perform worse, and sometimes much worse, than both our static hybrid and the McFarling predictor. In the case of the IBS suite, the PAs predictor does particularly badly because the IBS programs tend to have more branch sites than those in the SPEC suite. In the case of the SPECint95 benchmark geometric mean, it appears that the PAs predictors have a substantial advantage over the gshare predictors. The major contributors to the poor performance of gshare over SPECint95 are the programs “li” and “perl”. Both of these programs are interpreters and it is likely that a small number of important branches in the inner loop of the interpreter are better predicted with a local history mechanism such as provided by PAs. Each pass through the interpretation loop is likely to take a different path, corresponding to the byte-code being interpreted. Another feature of the two datasets presented is that they both include versions of gcc. In looking specifically at the results for gcc between the two benchmarks sets, we see that the relative performance of our static hybrid in comparison with the McFarling predictor is very comparable. This comparison supports the conclusion that the AB+BA cross-validation method we used for the IBS suite creates results that are comparable to those obtained from the separate train and ref cross-validation method used with the SPECint95 suite.

Larger Component Predictors Beyond the previous comparison, we investigated the performance of our static hybrid predictor and a McFarling predictor across a wide range of hardware sizes and geometries. The method for performing this comparison was as follows. We assume that both the McFarling predictor and our predictor have gshare and PAs components. Furthermore, we assume that the McFarling dynamic selector geometry has the same size and geometry as its gshare component. In the comparisons, we did include the McFarling selector in the number of bits the predictor requires. In our calculation of predictor size, we assume that BTB tags are free.

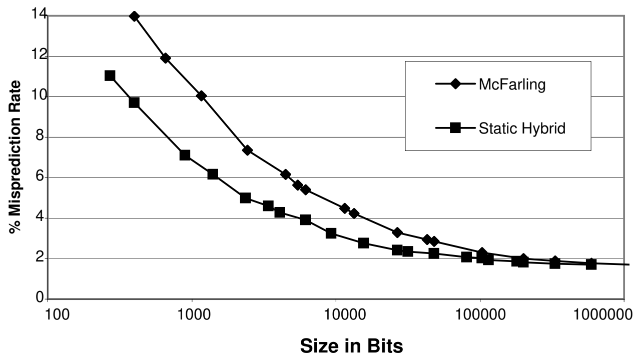


Figure 3. Various sizes of Static Hybrid versus McFarling

To calculate the optimal geometry (i.e., configurations of the gshare and PAs components in the hybrid) for a given hardware size is computationally very difficult because it involves searching a large four-dimensional space. To approximate the optimal geometry, we selected an initial set of hardware sizes and geometries that we felt were likely to be effective. We evaluated all of these configurations using the gcc benchmark in the IBS suite, and selected the most effective geometries for each hardware size. We then generated additional hardware configurations in the neighborhood of the most effective and iterated the process. This process was performed independently for the static hybrid and the McFarling predictors. As a result, the optimal geometry of a given size used for the McFarling results and Static Hybrid results is very likely different.

With the geometry that was most effective for the gcc application at each hardware size, we computed the geometric mean of the miss rates over the IBS benchmark suite, and these results are reported in Figure 3. The figure shows that the Static Hybrid predictor has substantially better performance than the McFarling predictor over a wide range of sizes, from very small to up to approximately 100,000 bits. Above that size, the performance of the predictors converges, as we would expect it to.

Comparison with Branch Classification We also investigated how a branch classification approach [2] might compare with the static hybrid we propose. With branch classification (specifically, the proposed PG+gshare technique), which also uses profiles, the branches that are observed to be statically predictable to a given threshold are predicted statically, while the rest are predicted dynamically. We used the same assignment threshold of 95% prediction accuracy used in [2]. Since one of their two components (the static hint) has no hardware cost, we made the gshare component used by the Branch Classification scheme twice as large as that used in our static-hybrid predictor. The geometric mean of the misprediction rate of such a configuration over the SPECint95 benchmarks is 4.85% compared to our misprediction rate of 3.10%.

This result can be explained in the following way. By selecting a classification threshold of 95% prediction accuracy, a static hint bit will be used for a large number of branches, and the miss rate for the branches predicted statically is likely to be around the threshold (i.e., a 5% misprediction rate) assuming that the branches were correctly predicted by the training set. Some statically predicted branches will have even lower miss rates, but at the same time some will have been incorrectly predicted by the training set, and thus have much higher miss rates. With a large fraction of branches incurring on average a 5% misprediction rate, then no matter how effective the gshare component is, the overall misprediction rate will remain relatively high.

Explanation of Performance Improvements To better understand the source of the performance improvement observed, we investigated several alternate configurations of our static hybrid predictor and the McFarling dynamic hybrid predictor. The results of this comparison, over the IBS benchmark suite, are presented in Figure 4. We present the results for IBS because we saw the greatest advantage of our static hybrid in this suite. The figure shows our static hybrid predictor (with partial update semantics, as described in Section 3), indicated by the "Partial Update" bars, the standard McFarling predictor, and two alternatives. The "Full Update" bars indicate the performance of our static hybrid predictor in an implementation in which both the gshare and PAs component predictors are both updated every time a branch resolves. The "Unbounded McFarling" bars indicate an implementation of the same McFarling predictor in which the dynamic selection hardware uses unbounded resources, and thus no aliasing occurs.

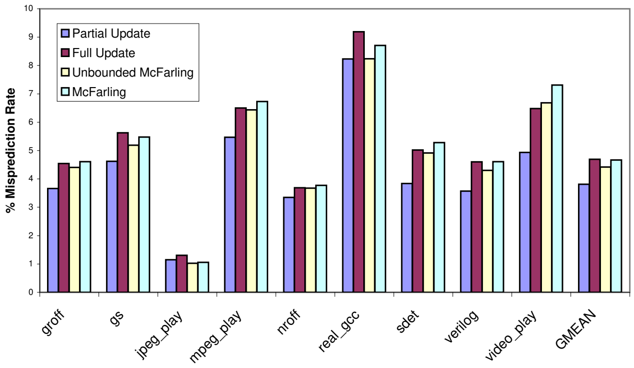


Figure 4. Performance Opportunities Available to Our Static Hybrid Predictor over IBS

The figure shows the two most important reasons that the static hybrid method is able to outperform the McFarling predictor. First, the "Full Update" static hybrid predictor results show a substantial performance degradation over the "Partial Update" predictor. This degradation occurs because the update of both components results in increased aliasing of the components and hence increased misprediction rates. It is difficult for a McFarling predictor to take advantage of a partial update semantics. Because the assignment of branches to components is done dynamically in the McFarling predictor, each of the two component predictors may be called upon at any time to predict a branch, thus they must both be kept fully updated.

The figure also shows that aliasing in the McFarling component selection hardware results in reduced performance for the McFarling predictor. While the effect is not as dramatic as the benefit from partial update semantics, the "Unbounded McFarling" performs uniformly better than the implementable McFarling predictor.

The greatest exposure of the static hybrid method in comparison to the McFarling predictor is that the decision about which component predictor to used is made statically, and hence could be incorrect a majority of the time. Our results indicate that the potential advantages of the static hybrid method outweigh the disadvantages when appropriate profile-based component selection is performed. In the next section, we explore what happens when an inappropriate profile-based training is performed.

Explanation of Performance Degradation The static hybrid predictor has worse performance than McFarling for two program, "go" and "li". Part of this performance difference can be explained by improper training in the profiling phase. Figure 5 provides information on the number of branch sites excerised by each input of the SPECint95 benchmarks. Each program has two bars associated that indicate the percentage of branches executed in the testing phase that were also executed in the training phase. The "Unseen with TRAIN/REF" bar assumes that the "train" input is used in training phase, and the "ref" input is used in the testing phase (the procedure used for all previous results shown). The "Unseen with REF/TRAIN" bar assumes that the "ref" input was used for training and the "train" input was used for testing.

Both "go" and "li" exhibit poor coverage in the training input set; in other words, the training input does not reflect the actual use of the application when the 'ref" input is used. Thus, the static hybrid method, like all profile-based branch predictors, is sensitive to training; however, unlike other methods, the *static metapredictor* is only used to determine which *dynamic* predictor to use, reducing the possible performance loss.

Understanding How the Prediction Components are Used To better understand how the different dynamic predictors were using their component predictors, we measured the "balance" between the components in the following way. For a particular input (e.g., "train" or "ref" from SPECint95) to a program, we ran each trace through both component predictors. We assigned each branch site in the program to the component that predicted that branch correctly most often (e.g., the standard procedure for assigning branches with our static hybrid predictor). We labeled branch sites where the "win" count for gshare was within 5 of that for PAs as "tie" branch sites.

Figures 6, 7, and 8 show the balance of the component predictors for our static predictor on the "train" SPECint95 input, our static predictor on the "ref" SPECint95 input, and the McFarling SPECint95 "ref" input, respectively. Figure 6 shows that the "train" input results in many branches being considered ties, which is likely caused by the fact that the inputs are sufficiently small that many branches are only executed a small number of times. In the cases of ties, using a static hybrid method, we have to choose one or the other of the component predictors anyway.

Comparing Figures 6 and 7, we see that the correct assignment of many of those tie branches would have been to assign them to the gshare component. Using gshare in the case of ties or no knowledge (e.g., the training input did not exercise the branch) is an intuitive choice in any event, because the gshare predictor is less perturbed by interference from multiple branch sites. This is partly because gshare allows aliasing, and hence should have graceful degradation compared to a tagged predictor.

Figure 8 shows the usage of each component when using the McFarling predictor over the SPECint95 "ref" dataset. There is no "tie" category since the McFarling predictor

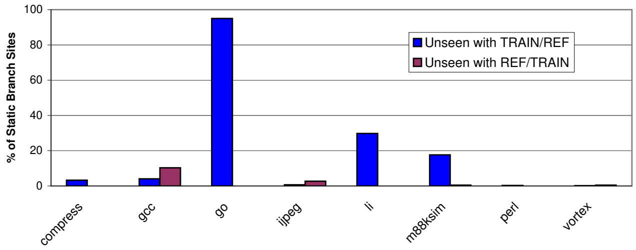


Figure 5. SPECint95 Branch Site Coverage

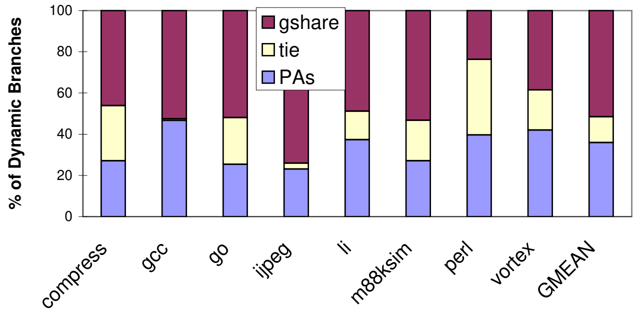


Figure 6. Balance of SPEC95 "train" as seen by static hybrid.

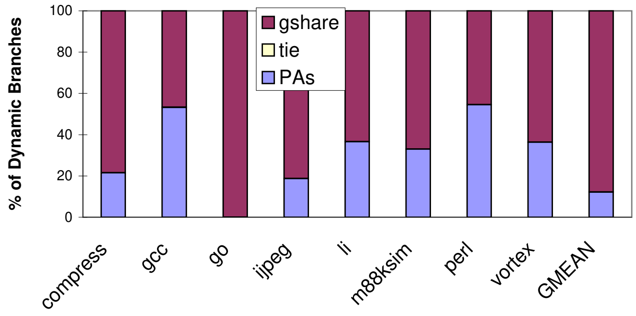


Figure 7. Balance of SPEC95 "ref" as seen by static hybrid.

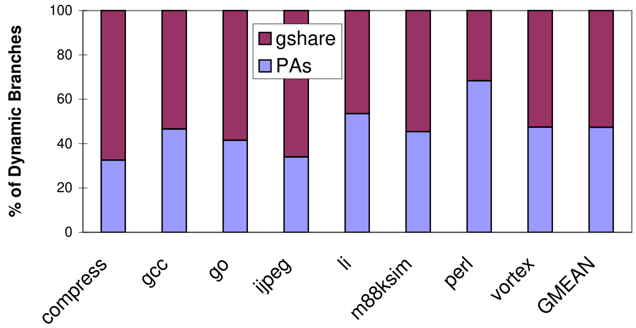


Figure 8. Balance of SPEC95 "ref" as seen by McFarling.

chooses one or the other component. The figure shows that the McFarling uses the PAs predictor overall more than our static hybrid predictor will, and in some cases, like go, substantially more. The very high miss rate in go for all predictors is likely to cause the McFarling predictor to switch between gshare and PAs somewhat randomly, resulting in more balance than is appropriate. The two programs in which PAs is used the most by the McFarling predictor are li and perl, both interpreters. As we have noted before, the PAs component appears more effective overall in these cases.

6 Conclusions

We have compare the performance of a *static vs. dynamic* meta-predictors in a hybrid branch predictor. In general, the static meta-predictor has better performance when suitable input profiles are used. All the results we presented were fully cross validated. Our static hybrid predictor appears unlikely to exhibit performance breakdowns, appears scalable, is able to cope with variations in the input sets, and does not require long training periods. Static meta-predictors also exhibit consistently good performance relative to well-regarded dynamic predictors. The performance was rarely worse than that of the more-expensive dynamic predictor, and was never worse on the more realistic IBS traces.

Acknowledgements: Donald Lindsay was supported by a gift from Hewlett-Packard. We would also like to thank Hewlett-Packard and Digital Equipment Corporation for an equipment grant that provided the simulation cycles and the anonymous referees for providing helpful comments. This work was partially supported by NSF grants No. CCR-9401689 and No. MIP-9706286.

References

[1] P.-Y. Chang, E. Hao, and Y. Patt. Alternative implementations of hybrid branch predictors. In *28th Intl. Symp. on Microarchitecture*, pages 252–257, 1995.

[2] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. Patt. Branch classification: a new mechanism for improving branch predictor performance. In *27th Intl. Symp. on Microarchitecture*, November 1994.

[3] Pohua Chang and Utpal Banerjee. Profile-guided multi-heuristic branch prediction. In *1995 Intl. Conf. on Parallel Processing*, pages 215–218, 1995.

[4] Thomas M. Conte, Burzin A. Patel, and J. Stan Cox. Using branch handling hardware to support profile driven optimization. In *27th Intl. Symp. on Microarchitecture*, November 1994.

[5] Jeffrey Dean, James E. Hicks, Carl A. Waldspurger, William E. Weihl, and George Chrysos. ProfileMe: Hardware support for instruction-level profiling on out-of-order processors. In *30th Annual Intl. Symp. on Microarchitecture*. IEEE, December 1997.

[6] Joel Emer and Nikolas Gloy. A language for describing predictors and its application to automatic synthesis. In *24th Annual Intl. Symp. of Computer Architecture*, pages 304–314, 1997.

[7] John H. Edmondson et al. Internal organization of the Alpha 21164, a 300 Mhz 64-bit quad-issue CMOS RISC microprocessor. *Digital Technical Journal*, 7(1):119–135, 1995.

[8] M. Evers, P.-Y. Chang, and Y. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In *23nd Annual Intl. Symp. of Computer Architecture*, pages 3–11, 1996.

[9] Linley Gwennap. Digital 21264 sets new standard. *Microprocessor Report*, 1997. Article available online at http://www.digital.com/semiconductor/microrep/digital2.htm.

[10] Chih-Chieh Lee, I-Cheng K. Chen, and Trevor N. Mudge. The bi-mode branch predictor. In *30th Intl. Symp. on Microarchitecture*. IEEE, 1997. (to appear).

[11] Scott McFarling. Combining branch predictors. Technical Report TN-36, Digital Equipment Corporation, Western Research Lab, June 1993.

[12] Pierre Michaud, Andre Seznec, and Richard Uhlig. Trading conflict and capacity aliasing in conditional branch predictors. In *24th Annual Intl. Symp. of Computer Architecture*, pages 292–303, 1997.

[13] Parthasarathy Ranganathan and Norman Jouppi. The relative impact of memory latency, bandwidth and branch limits to microprosssor performance. Presented at 1st. Workshop on Mixing Logic and DRAM: Chips that Compute and Remember, held in conjunction with the 1997 Intl. Symp. on Computer Architecture, Denver, Colorado, June 1997.

[14] S. Sechrest, C.-C. Lee, and T. Mudge. Correlation and aliasing in dynamic branch predictors. In *23nd Annual Intl. Symp. of Computer Architecture*, pages 22–32, 1996.

[15] Amitabh Srivastava and Alan Eustace. ATOM: A system for building customized program analysis tools. In *Proceedings of the SIGPLAN'94 Conf. on Programming Language Design and Implementation*, pages 196–205. ACM, 1994.

[16] Standard Performance Evaluation Corportation, Manassas, VA. *SPEC95 Technical Manual*, August 1995.

[17] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Instruction fetching: Coping with code bloat. In *22nd Annual Intl. Symp. of Computer Architecture*, June 1995.

[18] Kenneth C. Yeager. The MIPS R10000 superscalar processor. *IEEE Micro*, pages 28–40, April 1996.

[19] Tse-Yu Yeh and Yale N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Proc. 19th Annual Symp. on Computer Architecture*, pages 124–134, May 1992.