Proceedings

International Conference on Computer Design

VLSI in Computers and Processors

October 12-15, 1997

Austin, Texas

Sponsored by

IEEE Computer Society Technical Committee on Design Automation

IEEE Circuits and Systems Society

Design Optimization for High-speed Per-address Two-level Branch Predictors

I-Cheng K. Chen, Chih-Chieh Lee, Matt Postiff, and Trevor Mudge

EECS Department, University of Michigan

1301 Beal Ave., Ann Arbor, Michigan 48109-2122

{icheng, leecc, postiffm, tnm}@eecs.umich.edu

Abstract

Per-address two-level branch predictors have been shown to be among the best predictors and have been implemented in current microprocessors. However, as the cycle time of modern microprocessors continues to decrease, the implementation of set-associative per-address two-level branch predictors will become more difficult. Instead, direct-mapped designs may be more attractive. In this paper, we investigate an alternative implementation of the per-address two-level predictor referred to as the tagless, direct-mapped predictor, which is simpler and has faster access time. The tagless predictor can offer comparable performance to current set-associative designs since removal of tags allows more resources to be allocated for the predictor and branch target buffer (BTB). Removal of tags also decouples the per-address predictors from the BTB, thus allowing the two components to be optimized individually. Furthermore, our results show that this tagless implementation is more accurate because it handles conflict misses in the branch history table better.

Finally, we examine the system cost-benefit for tagless per-address predictors across a wide design space using equal-cost contours. We study the sensitivity of performance to the workloads by comparing results from the Instruction Benchmark Suite (IBS) and SPEC CINT95. Our work provides principles and quantitative parameters for optimal configurations of such predictors.

1. Introduction

As microprocessor designs move toward wider instruction issue and deeper pipelines, effective branch prediction becomes essential to exploiting full performance. A good branch prediction scheme increases the performance of a microprocessor by eliminating instruction fetch stalls in the pipelines. As a result, numerous high performance branch prediction schemes have been proposed, such as two-level adaptive branch predictors [Yeh91], correlation-based predictors [Pan92], and hybrid branch predictors [McFarling93, Chang94].

Among different predictors proposed, the per-address two-level branch predictor has been shown to be one of the best and has been implemented in the Intel Pentium Pro processor [MReport95]. Typically, the two-level per-address predictor is coupled with a branch-target buffer (BTB) through the sharing of common tags [Yeh92, Calder94]. Both components benefit from tags and, thus, cost can be reduced by sharing. In particular, the tags enable high hit-rate set-associative design for the predictor and the BTB.

However, as the clock frequency of modern microprocessors continues to increase, the coupled set-associative design using tags may no longer be the best choice. This is because set-associative designs require longer access time than direct-mapped designs and, thus, may become a critical path in a high clock rate microprocessor. Therefore, we re-evaluate an alternative tagless direct-mapped version of two-level per-address predictors [Yeh91].

A tagless direct-mapped per-address predictor can offer performance comparable to current implementations. Typically the tagless predictor does not have hit-rates that are as high as a set-associative design, but it offers two advantages. First, by removing tag storage, more resources can be allocated to the predictor and BTB to improve performance. Second, by decoupling the BTB from the predictor, the tagless design offers the flexibility to optimize the BTB and predictor individually. In particular, the predictor can have a different number of entries than the BTB. Thus, the BTB need only store taken branches instead of all branches [Smith81, Calder94]. Also note that the removal of tags does not prevent the identification of branch instructions, because branches can still be identified using predecode information, which is already commonly employed in commercial microprocessors.

To justify the tagless implementation, we conduct performance evaluation and show that, for the prediction process, tagless predictors in general perform better, or no worse, than direct-mapped tagged predictors. To analyze the improvement, we break down the total errors into transitional-state and steady-state errors. We found that tagless predictors have lower transitional errors and, consequently, have higher performance. Moreover, the tagless predictor is simpler and faster than the tagged version.
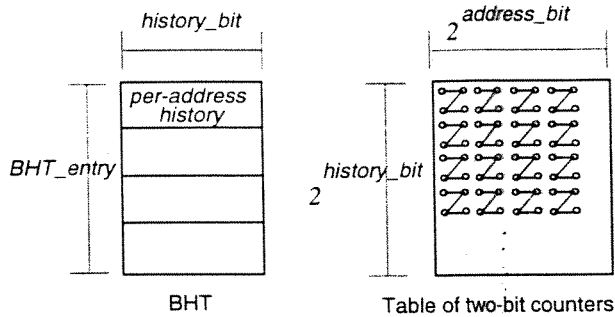
Figure 1: Schematic for a per-address two-level branch predictor

To develop general design principles for optimal configurations, we exhaustively search the design space of tagless per-address predictors. Our study shows the sensitivity of the optimal configurations to various program characteristics. To conclude, we derive general principles for selecting the best parameters. When given a specific budget and benchmark suite, these principles can help designers to select the best configurations.

The rest of this paper is organized as follows. In Section 2 we briefly review the per-address two-level predictor, and discuss the tagless per-address prediction scheme. In Section 3 we explain why the tagless scheme can have a better prediction accuracy than a traditional tagged scheme. Section 4 develops a cost analysis procedure to identify optimal tagless predictor designs. We present some concluding remarks with Section 5.

2. Per-address two-level branch predictors

The two-level per-address adaptive branch predictor is a variation of two-level branch predictors proposed by Yeh and Patt [Yeh92, Yeh93]. As shown in Figure 1, a two-level per-address adaptive branch predictor consists of two tables. The first-level table, called the branch history table (BHT), has multiple shift-registers called branch history registers (BHRs). Each of these registers is used to record past branch outcomes for a single static branch. The branch outcome patterns recorded in the first-level table are then used to index a set of counters in the second level. The column index into the counters is usually some part of the address of the branch being predicted. Although there are many options for the counters, the best performance has been observed when the counters are two-bit saturating up-down counters [Nair95, Chen96a].

Since the counters are typically organized as a two-dimensional array, there can be many configurations for the second-level table. If a configuration has multiple rows and columns, then it is generally referred to as a PAs scheme according to the taxonomy by Yeh and Patt [Yeh93]. If the table has a single column. it is a PAg scheme. If the table is a single row, the predictor is equivalent to the traditional two-
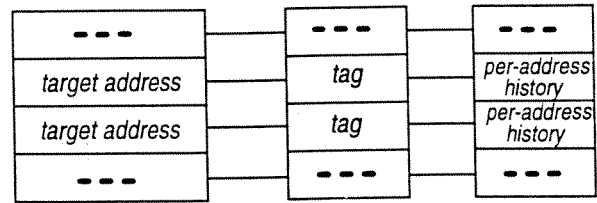

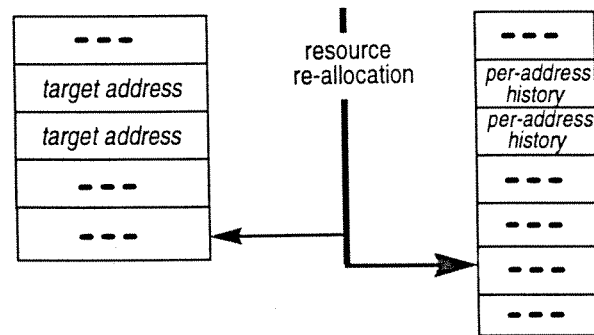**Figure 2a: Tagged per-address two-level branch predictor**

Figure 2b: Tagless per-address two-level branch predictor

bit counter scheme proposed by Smith [Smith81] because the counters are exclusively indexed by the branch address. This design space has been thoroughly studied by Sechrest et al. [Sechrest96].

The two-level per-address predictor has been shown to be among the best predictors currently in use. It has also been adopted in industry for high performance microprocessors. For example, the Intel Pentium Pro employs a two-level per-address predictor with a 512-entry 4-way BHT, where each BHT entry records 4-bit per-address history [MReport95].

2.1 Tagless implementation

In a typical two-level per-address scheme, the predictor is coupled with a branch target buffer (BTB) through the sharing of common tags [Yeh92, Calder94], as shown in Figure 2a. This coupling of predictor and BTB is cost effective since the predictor and BTB share a single copy of the tags. The presence of tags also allows set-associative predictors, which provide a high hit rate for both predictor and BTB.

Unfortunately, as the cycle time of microprocessors continues to decrease, the coupled-set-associative design using tags may no longer be the best choice. Set-associative implementations require longer access time than direct-mapped designs, and may become a critical path in a microprocessor with short cycle time. In addition, the coupling of
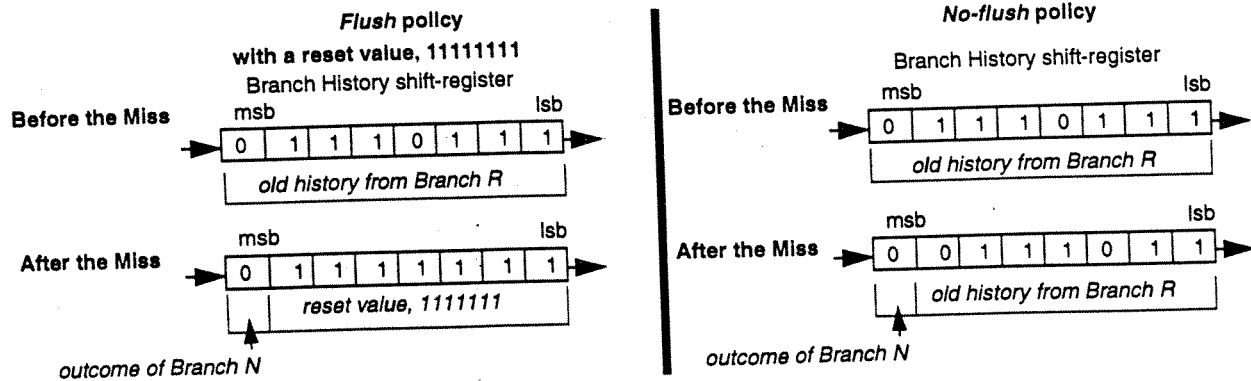
89

Flush policy
with a reset value, 11111111
Branch History shift-register

Before the Miss
msb / lsb
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
old history from Branch R

After the Miss
msb / lsb
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
reset value, 1111111
outcome of Branch N

No-flush policy
Branch History shift-register

Before the Miss
msb / lsb
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
old history from Branch R

After the Miss
msb / lsb
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
old history from Branch R
outcome of Branch N

Figure 3: An example comparing the *flush* and *no-flush* policies

This example illustrates the difference between the *flush* and *no-flush* miss handling policies for a conflict miss in the branch history table (BHT). In this example, each BHT entry, i.e. the branch history shift-register, can store an 8-bit per-address history of the branches that index it; 0 represents not-taken and 1 represents taken. R represents an old branch being replaced, and N represents the new incoming branch. As illustrated, the incoming branch, N, happens to index the same shift-register that R does, so a miss occurs. The *flush* policy changes the register content to all 1's and shifts in the new outcome, which is 0, after outcome of N is resolved. In contrast, the *no-flush* policy simply shifts in the new outcome, 0, without flushing the old history of R.

predictor and BTB degrades performance in two ways. First, the BTB needs to allocate space to record not-taken branches, since the predictor needs information for all branches. This wastes BTB resources [Calder94]. Second, the number of history entries in the predictor is limited to be the same as the number of BTB entries, restricting the designer's freedom to fully explore the design space. As cycle times become shorter, we believe that decoupled, direct-mapped per-address schemes, as shown in Figure 2b, deserve closer inspection.

Note that the removal of tags does not prevent the identification of branch instructions. Branch instructions can still be identified using predecoded information stored in the cache, which is already commonly employed in commercial microprocessors.

In a decoupled, direct-mapped per-address scheme design, tags for the branch predictor are redundant. Tags are crucial for a set-associative BHT to distinguish branches in the same set. In contrast, in a direct-mapped BHT, no such distinction is needed, so tags only affect the miss handling policy. More specifically, if there is a miss (or conflict), the predictor needs to decide whether to use the history from old branch, or flush the history register and restart with some predefined "reset value." The former scheme does not need tags at all, and we refer it as the tagless branch predictor in this paper. As an aside, we note that the tagless implementation of per-address predictors can be categorized as a per-set history scheme, according to [Yeh93], because several branches may share one history register.

In the next section, we will show that, because of better miss handling policy, the tagless scheme is actually superior to the direct-mapped tagged predictor.

3. Performance analysis for the tagless predictors

In this section, we investigate how tags affect the prediction mechanism, excluding the hit rate factor. Tags have no effect when a branch hits in the history shift-register; tags only affect prediction accuracy when misses or conflicts occur. Tagged schemes can employ different miss handling policies which in turn can yield different prediction accuracies.

3.1 Miss handling policies

When misses occur, the branch history shift-register has the option of flushing or not flushing its old history contents, as shown in Figure 3. Tagless predictors must employ the *no-flush* policy since they do not have tags to detect a "miss." Tagged predictors, on the other hand, can implement the *flush* policy, where the history is flushed and reset to a default *reset value*. The old history is discarded, and the incoming branch starts accumulating its own history. This miss handling policy has intuitive meaning and takes advantage of the tags.

3.2 Simulation Methodology

To fairly compare the tagged and the tagless predictors, the best tagged predictor must be determined and used for comparison. We exhaustively simulated all possible 256 reset values for 8-bit history patterns and sampled 256 reset values for 14-bit predictors to find the best reset value for

Benchmarks		static conditional branches	dynamic conditional branches
SPEC95	compress	95	10,216,264
	gcc	15,647	24,048,361
	go	4,742	18,168,554
	ijpeg	902	40,854,598
	li	345	24,977,690
	perl	1,576	31,309,305
	vortex	5,963	24,979,201
IBS	groff	6,333	11,901,481
	gs	12,852	16,307,247
	mpeg_play	5,598	9,566,290
	nroff	5,249	22,574,884
	real_gcc	17,361	14,309,867
	sdet	5,310	5,514,439
	verilog	4,636	6,212,381
	video_play	4,606	5,759,231

Table 1: Static and dynamic conditional branch counts in SPEC CINT95 and IBS programs

each benchmark. Since the best reset value is different for each benchmark, we present results using the best value for each benchmark[1], instead of applying one common value for all. This creates an unreachable upper bound for the performance of the tagged predictor.

To focus on the prediction mechanism, we use direct-mapped schemes for both tagless and tagged predictors to isolate the effect of hit rate. For simplicity, we select PAg predictors of 8-bit and 14-bit history for our comparisons.

To assess the performance of tagless and tagged predictors, we conduct a trace-driven simulation. As input for the simulation, we use the Instruction Benchmark Suite (IBS) benchmarks [Uhlig95] and the SPEC CINT95 benchmarks [SPEC95]. The branch statistics for both benchmark suites are summarized in Table 1.

The IBS benchmarks are a set of applications designed to reflect realistic workloads. The traces of these benchmarks are generated through hardware monitoring of a MIPS R2000-based workstation. These traces were collected under Ultrix 3.1 and include both kernel-level and user-level addresses.

For the SPEC CINT95 benchmark, we used ATOM [Eustace95], a code instrumentation interface from Digital Equipment Corporation, to generate address traces. The benchmarks are first instrumented with ATOM, then executed on a DEC 21064 workstation running OSF/1 3.0 to generate traces. These traces contain only user-level instructions.[2]

---

1. As an aside, we noticed that the best reset value is the least frequently occurring history pattern in the benchmark, because it causes least interference in the normal prediction process. This "best" reset value differs for each benchmark.

2. Input to the SPEC95 benchmarks was a reduced input data set; each benchmark was run to completion.

3.3 Simulation results

Figure 4 shows the average misprediction rates for tagless and tagged predictors for the IBS benchmarks. PAg predictors with 8-bit history are used in this simulation. Since the y-axis represents the misprediction rate, a lower bar indicates better performance. The x-axis represents the number of history shift-registers. Within each pair of bars, the left bar represents the tagged predictor, and the right bar represents the tagless predictor. The meaning of the gray and dark components will be explained in section 3.4. As shown in Figure 4, the tagless predictor outperforms the upper bound of tagged predictor in a 128-entry branch history table, while both predictors have similar misprediction rates in 1K- and 4K-entry configurations. Similar conclusions can be drawn for SPEC CINT95 benchmarks, as Figure 5 shows.

When the length of history is increased from 8 bits to 14 bits, the misprediction rates for IBS are shown in Figure 6. In this case, the tagless predictor outperforms the upper bound of the tagged predictor for both 128 and 1K history shift-registers configurations, and they perform very closely to each other in 4k configuration. Similar conclusions can be drawn for SPEC CINT95 as Figure 7 shows.

Although we presented the averaged results, the tagless predictor is better than the tagged predictors in almost all the benchmarks. Detailed data for each benchmark can be found in [Chen96b].

In conclusion, the tagless predictors perform better when the number of entries in branch history table is small or when history length is long; tagless predictors have comparable performance as tagged predictors in other configurations. Furthermore, tagless predictors are both simpler and cheaper.

3.4 Analysis using transitional-state and steady-state error

To explain the superior performance of tagless predictors, we broke down the total error into transitional-state error (black portion of the bars), and steady-state error (gray portion of the bars), shown in Figures 4 to 7. Since tagless and tagged predictors differ in the miss handling policy (*flush* or *no-flush*), which affects the transitional state, we classify the error into these two categories to identify the sources of prediction error.

At any time, each of the history shift-registers, i.e. each BHT entry, is either in a transitional state or in a steady state. In a transitional state, the history of a branch does not fill up the entire history shift-register. In other words, only part of the history information belongs to the current branch and the rest of the history information is either part of the reset value (flush policy), or history left from the replaced branch (no-flush policy). The transitional state occurs on the first few references right after a miss, and is, in a sense,
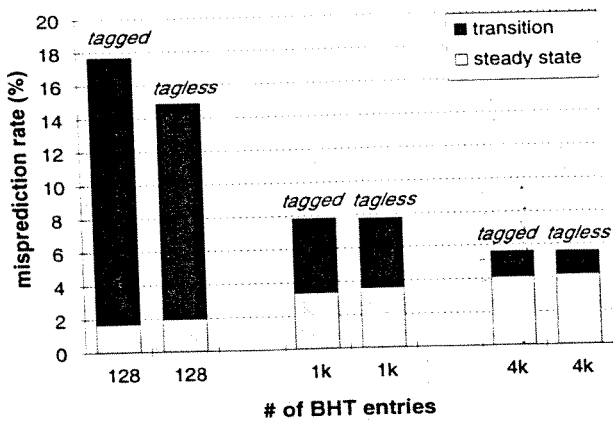
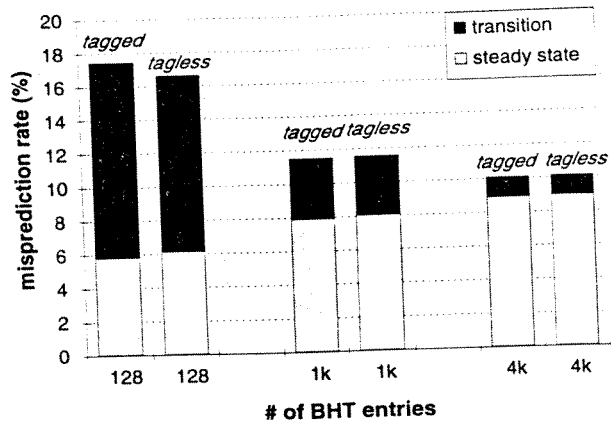**Figure 4: Misprediction rate of PAg with 8-bit history length for IBS**

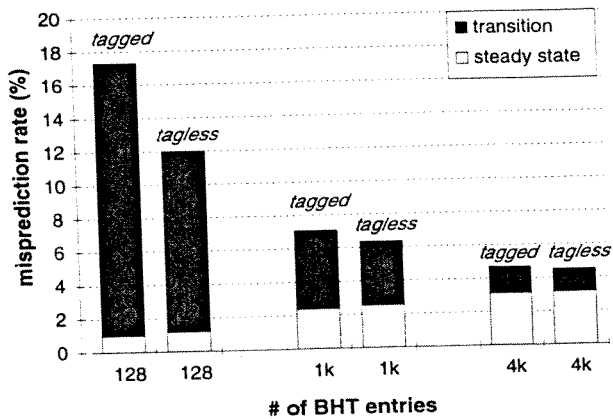**Figure 5: Misprediction rate of PAg with 8-bit history length for SPEC CINT95**

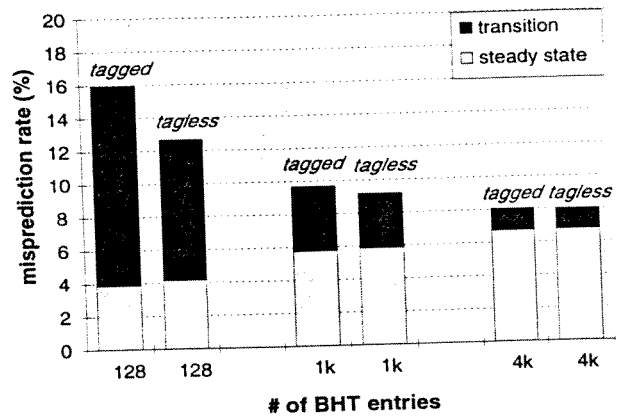**Figure 6: Misprediction rate of PAg with 14-bit history length for IBS**

Figure 7: Misprediction rate of PAg with 14-bit history length for SPEC CINT95

similar to "cold starts" for caches. On the other hand, the steady state is reached when a history shift-register is filled up with branch outcomes exclusively from the current branch.

For example, an 8-bit branch history shift-register is in a transitional state during the first 8 references right after a miss, since it takes 8 references (branch outcomes) to update and fill up the history shift-register. From then until the next miss occurs to the same shift register, the branch history shift register is in its steady state.

We observed that the tagless predictors have less transitional error than the tagged predictor. The transitional-state error is due to the partially correct history which is likely to index to a wrong 2-bit counter, resulting in incorrect prediction. As can be seen in Figures 4 to 7, the transitional-state errors for tagless predictors are smaller for configurations

with a small branch history table (128 entries), and long history length configurations (14 bits).

To explain why tagless predictors have smaller transitional errors, we examine the miss handling process of tagged predictor. Every time a miss occurs, a tagged predictor flushes the old contents of the history shift-register, resets it to a default value, then starts accumulating the history information from the new branch. However, if misses occur too often, the contents of the history shift-register will be flushed constantly and, thus, never reach a steady state. In this case, most of the transitional state predictions only use the 2-bit counter indexed by the reset value, resulting in large transitional-state error. This situation gets worse when history length is long, because it takes longer to reach steady state.
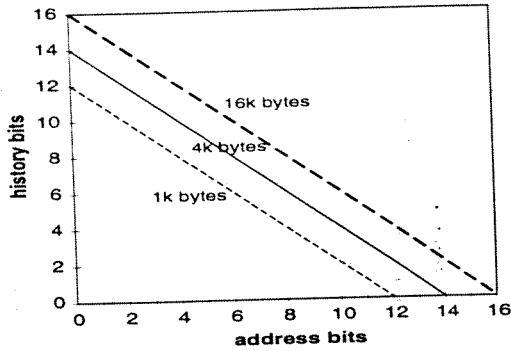
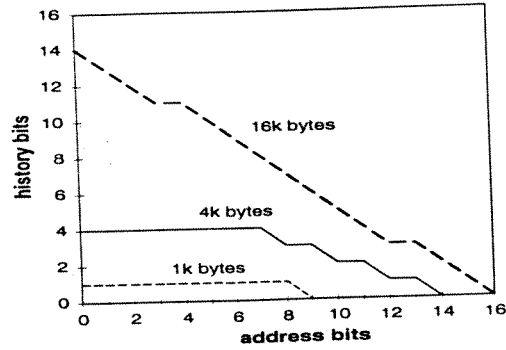**Figure 8: Equal-cost contours for 128 branch history entries**

Figure 9: Equal-cost contours for 8k branch history entries

In contrast, a tagless predictor does not flush old history when misses occur. On a miss, depending on the characteristics of the previous branch, old history information may not always be harmful. For example, if a mostly-taken branch (e.g., loop branch) is replaced by another mostly-taken branch, the old history information would be the same as that of new coming branch and, thus, resulting accurate prediction. In another situation, consider the case that a frequently occurring branch X is only briefly interrupted by another branch Y. When the branch X comes back, most of the old history information would still belong to X, which helps the prediction. This also explains why tagless predictors perform better when history length is long.

The steady-state error is essentially independent of the miss handling policy, and hence prediction accuracy for steady state should be almost the same for both tagless and tagged predictors. Indeed, the steady-state errors (shown as the lower gray bars in Figures 4 to 7) are about the same for both predictors. The prediction accuracy during the steady state is high, because all the history used for prediction belongs to the current predicting branch. Therefore, the difference between tagless and tagged schemes lies in the error during the transitional state.

In summary, tagless predictors can have better overall results than tagged predictors, in addition to its simpler implementations with more design flexibility.

4. Cost-benefit analysis for tagless predictors

After having shown the effectiveness of tagless per-address two-level predictors, we present a cost-benefit analysis for a wide range of configurations in its design space. There have been some previous studies for per-address schemes [Yeh93, Sechrest96]. However, their work mainly focused on the design trade-off for the second-level table, while we incorporate the first-level table cost for a complete analysis. We examine hardware budgets ranging from 512 bytes to 16K bytes. The three parameters considered in our design space are: the number of entries in branch history ta-

ble (BHT), the number of address bits indexing 2-bit counters, and the number of history bits in the branch history registers. These three parameters are labeled as *BHT_entry*, *address_bit*, and *history_bit*, respectively; see Figure 1 for a pictorial representation. The estimated cost in bits for the tagless per-address scheme is given as follows:

$$\text{Cost} = (BHT\_entry) * (history\_bit) +$$
$$2^{(history\_bit + address\_bit + 1)}$$

Based on this cost function, we can derive equal-cost contour lines for a fixed number of branch history entries in the BHT. Note that history bits are inherently more expensive than address bits, because history bits require extra resources (BHT) to record them. Figure 8 and 9 show the equal-cost contour lines for 128 and 8K BHT entries respectively.

In Figure 8, equal-cost contours are diagonal straight lines. This implies that, when the budget is fixed and the BHT entries are few, substituting one address bit with one history bit will incur almost no extra cost. In other words, the costs of each history bit and address bit are almost equal.

However, when the number of BHT entries is large, as shown in Figure 9, the equal-cost contours are almost parallel to the x-axis for small budgets. This implies that with the same budget, we can have more address bit than history bits. This is because when the number of BHT entries is large, the cost of each additional address bit is insignificant to that of each additional history bit.

4.1 Cost/performance analysis

Figure 10 shows the optimal points for different budgets and configurations for the SPEC CINT95 benchmarks. Various configurations with the same budget are grouped as a clusters of bars, where each bar represents the best point for a fixed number of BHT entries. The two-bit counter scheme is shown as a line. The best per-address two-level predictor consistently outperforms the 2-bit counter scheme.

However, for the IBS benchmarks (Figure 11), the two-bit counter scheme outperforms the per-address two-level
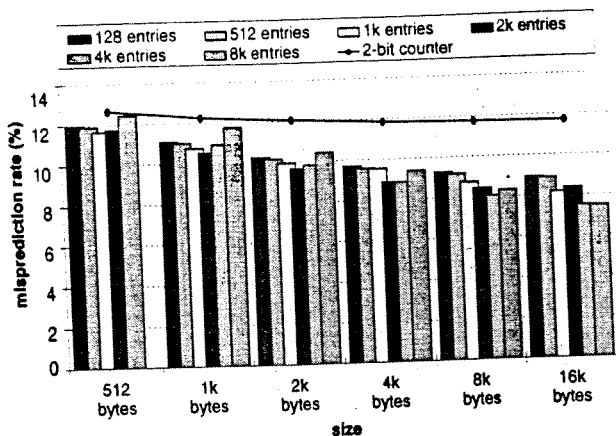
Figure 10: Misprediction rate vs. budget for SPEC CINT95

Figure 11: Misprediction rate vs.budget for IBS

predictor with small budget (512 bytes). This is because IBS have relatively large number of static branches to be distinguished and predicted. These branches can be distinguished and predicted using either address bits or history bits (patterns). History bits require extra resources (BHT) to record them, while address bits can be obtained from the program counter and, thus, are essentially free. Consequently, when the budget is small, history information cannot effectively distinguish large number of static branches, because there are not enough resources to build a large BHT to record history information. In this particular case, the two-bit counter scheme can outperform a two-level predictor, because it does not need any history. However, as budgets increase, the two-bit counter scheme improves only a little. In contrast, the per-address two-level predictor improves and outperforms the two-bit counter scheme.

To study the optimal designs for two-level predictors, we plot the misprediction rates for different budgets and numbers of BHT entries, shown in Figure 12 and 13. The *x*-axis indicates the number of entries in the BHT, and the *y*-axis indicates the misprediction rate. The dotted horizontal lines represent two-bit counter schemes (2bc). Each curve line in the figure represents a fixed budget, ranging from 512 bytes to 16k bytes. In addition to the number of BHT entries labeled in the *x*-axis, the optimal configuration for each budget is labeled with its two other parameters, formatted as (history bits, address bits).

When the budget is small, the address bits are very important. This is because, as previously explained, address bits are cheaper than history bits since they do not need extra storage resources. Therefore, when the budget is small, it is relatively more efficient to use address bits instead of history bits. This fact can be verified for SPEC, shown in Figure 12. Labeled as the right number in each ordered pair, the number of address bits for the optimal configuration is
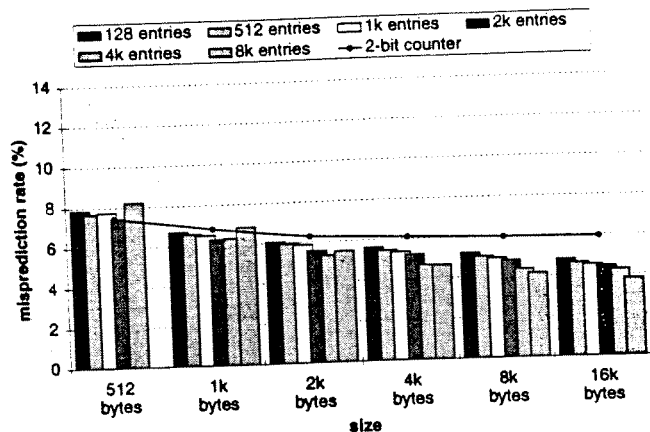
relatively large when the budget is small. IBS exhibits similar behavior, as shown in Figure 13. The only difference is that the number of address bits is even larger for IBS, since IBS has more static branches to be distinguished than the SPEC benchmarks.

However, as budgets increase, the importance of address bits rapidly decreases and is replaced by history bits. At first, it may seem to be counter-intuitive that, when budgets increase, the number of address bits decreases, instead of increasing correspondingly or at least remaining constant. This is because the increase in history bits (patterns) can also replace the function of address bits, which is to distinguish branches. Therefore, as the budget increases, the history bits gradually takes place of address bits and become more important. This trend can be seen for both SPEC and IBS, shown in Figure 12 and 13. Note that the number of address bits in SPEC decreases more rapidly because they contain less static branches.

The importance of history bits quickly becomes dominant as budgets increase. In addition to distinguishing static branches like address bits, history bits can distinguish and better predict different patterns within the same branch. Thus, history bits can offer additional benefit over the address bits. This benefit is particularly important for branches that are hard to predict. Moreover, this benefit becomes more significant as budgets increase, since more resources can then be allocated to record history information. This trend of increasing address bits can be verified in both the SPEC and IBS benchmarks, shown in Figure 12 and 13. As the optimal points move downward (budgets increase), the number of history bits increases (left numbers in the parentheses). We also notice that the number of history bits increase faster for SPEC, because branches in SPEC are relatively harder to predict [Sechrest96].
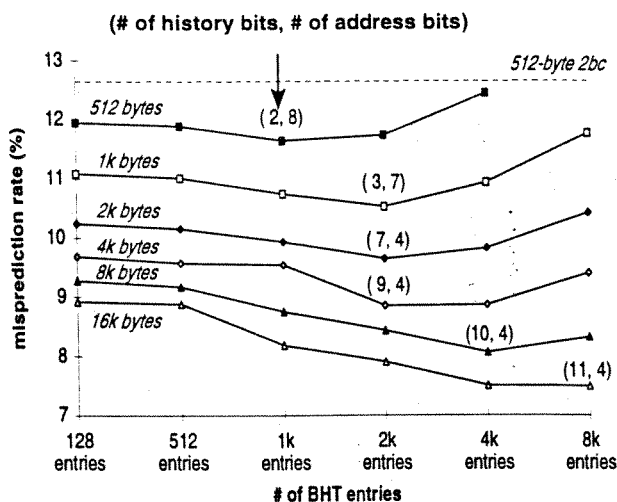
Figure 12: The optimal configuration for each budget in SPEC CINT95. Each optimal configuration is labeled with the number of history bits and address bits.
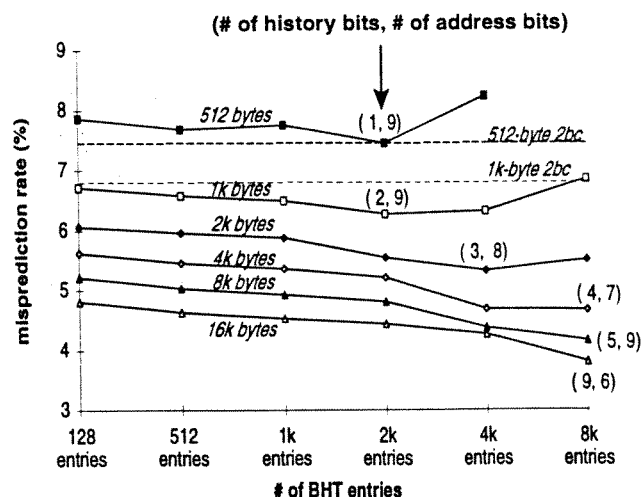
Figure 13: The optimal configuration for each budget in IBS. Each optimal configuration is labeled with the number of history bits and address bits.

Also note that as budgets increase, the number of entries in the branch history table also increases with the number of history bits. This trend can be explained by the increasing importance of history information. As described in the previous paragraph, history information becomes important when the budget is large. However, to effectively increase history information, increasing the number of history bits alone is not enough. Although by adding more history bits, more history information can be recorded in a longer branch history register, this information can easily be replaced and lost due to misses. Therefore, the number of entries in the BHT must be increased accordingly to improve the hit-rate. Since the number of BHT entries affects the hit rate, the number of entries needed is mostly determined by the number of static branches in the benchmarks. To see this trend, notice that the optimal configuration gradually moves toward the right along the *x*-axis (more entries in branch history table) as they move downward from one curve to another (greater budget), as shown in Figure 12 and 13.

As an aside, throughout the design budgets we examined, ranging from 512 bytes to 16KB, the tagless PAg scheme was never an optimal configuration. The tagless PAg is not cost-effective because it only uses expensive history bits, instead of relatively cheap address bits.

4.2 Design principles

The principles for designing tagless per-address predictors can be summarized as follows. When the budget is small, address information should be emphasized first. In other words, the number of address bits should be much larger than history bits (e.g., 8 address bits versus only 2 history bits). The number of address bits is determined by and

proportional to the number of static branches in the benchmarks.

As the budget increases, address bits should decrease accordingly; at the same time, resources should be allocated for more history bits as well as the number of branch history entries. Concerning the rate at which address bits should be replaced with history bits: more aggressive replacement should be adopted when branches are hard to predict, or when the number of static branches is small. For example, when the budget is small, the SPEC benchmarks achieves the best performance when the number of address bits is 4 times the number of history bits. However, when the budget is large, the best performance is achieved when the number of address bits is reduced to less than half the number of history bits.

In addition to increasing with the budget, the number of branch history entries is determined by the number of static branches in the benchmarks. A large number of branch history entries should be allocated if the number of static branches is large, e.g., 4k to 8k entries are needed for the benchmarks we examined.

When the numeric values for design parameters are needed, computer architects can first measure the statistics of the targeted benchmarks, which including the number of static branches and the misprediction rate for a baseline predictor within the budget. Then, to fine-tune the design parameters, these statistics can be compared to those of IBS and SPEC CINT95, as shown in Table 1, Figure 12, and Figure 13. Depending on how close the statistics are to those of IBS and SPEC, the parameters we have provided can either give a good estimate or, at least, significantly reduce the design space.

5. Concluding remarks

In this paper, we first evaluated the benefits of tagless per-address two-level branch predictors and then examined the design principles and cost/performance trade-off for a system with such a predictor.

To illustrate the benefits of tagless per-address predictors, we argued that they have faster access time, lower power, and simpler implementation than tagged predictors. At the same time, tagless predictors can offer performance comparable to the traditional tagged predictors by allowing additional resources to be allocated to the predictor and BTB and allowing these two components to be optimized as separate entities.

We also showed that the prediction accuracy of tagless predictors is better than direct-mapped tagged predictors. By characterizing the sources of prediction errors, we demonstrated the tagless predictor outperforms the direct-mapped tagged predictor due to the better capability of handling transitional-state branches.

We further evaluated cost and performance trade-off across a wide range of the design space. Equal-cost contour is the criteria for determining the best configuration. Based on the simulation results from the SPEC CINT95 and IBS benchmarks, we concluded that the number of address bits indexing into the second level table is the most important parameter when the available budget is small (e.g., 8 bit address bits versus only 2 history bits). However, the importance of address bits quickly diminishes as the budget increases. With a larger budget, history bits and the number of branch history entries should increase accordingly, but the number of address bits should be reduced. In addition, we noticed that the PAg scheme is never an optimal configuration over the budgets and configurations we examined.

Finally, we present a set of design principles for tagless per-address two-level predictors. First, we can measure the statistics of target benchmarks, which include the number of static branches and the misprediction rate for a base configuration. Then, we can compare these statistics with those from IBS and SPEC. The quantitative data collected from IBS and SPEC can provide a rough idea of how an optimal implementation should be. By carefully examining the interaction among different parameters, we also outlined the principles on how to fine-tune these parameters for better design.

Acknowledgment

References

[Calder94]   Calder, B. and Grunwald, Dirk. *Fast & accurate instruction fetch and branch prediction*. Proc. 21st Int. Symp. Comp. Arch., 1994.

[Chang94]   Chang, P., Hao, E., Yeh, T. and Patt, Y. *Branch classification: a new mechanism for improving branch predictor performance*. Proc. 27th Int. Symp. Microarchitecture, Nov. 1994.

[Chen96a]   Chen, I-C. K., Coffey, J.T. and Mudge T. *Analysis of branch prediction via data compression*. Proc. 7th Int. Conf. Arch. Support for Prog. Lang. and Op. Sys., Oct. 1996.

[Chen96b]   Chen, I-C. K., Lee, C-C., Postiff M. and Mudge T. Tagless two-level branch prediction schemes. Tech. Report CSE-TR-306-96, University of Michigan, 1996.

[Eustace95]   Eustace, A. and Srivastava, A. *ATOM: A flexible interface for building high performance program analysis tools*. Proc. Winter 1995 USENIX Technical Conf. UNIX and Adv. Comp. Sys., Jan. 1995.

[McFarling93] McFarling, S. *Combining branch predictors*. WRL Tech. Note TN-36, June 1993.

[MReport95]   Microprocessor Report, Sebastopol, CA: MicroDesign Resources, March 1995.

[Nair95]   Nair, R. *Optimal 2-bit branch predictors*. IEEE Trans. on Computers, Vol. 44, No. 5, May 1995.

[Pan92]   Pan, S-T., So., K. and Rahmeh, J.T. *Improving the accuracy of dynamic branch prediction using branch correlation*. Proc. 5th Int. Conf. on Arch. Support for Prog. Lang. and Op. Sys., 1992.

[Sechrest96]   Sechrest, S., Lee, C-C. and Mudge, T. *Correlation and aliasing in dynamic branch predictors*. Proc. 23rd Int. Symp. on Comp. Arch., May 1996.

[Smith81]   Smith, J.E. *A study of branch prediction strategies*. Proc. 8th Int. Symp. Comp. Arch., May 1981.

[SPEC95]   SPEC CPU'95, Technical Manual, August 1995.

[Uhlig95]   Uhlig, R., Nagle, D., Mudge, T., Sechrest, S. and Emer, J. *Instruction Fetching: Coping with Code Bloat*. Proc. 22nd Int. Symp. Comp. Arch., June 1995.

[Yeh91]   Yeh, T-Y. and Patt, Y. *Two-level adaptive training branch prediction*. Proc. 24th Int. Symp. Microarchitecture, Nov. 1991.

[Yeh92]   Yeh, T-Y. and Patt, Y. *A comprehensive instruction fetch mechanism for a processor supporting speculative execution*. Proc. 25th Int. Symp. Microarchitecture, December 1992.

[Yeh93]   Yeh, T-Y. and Patt, Y. *A comparison of dynamic branch predictors that use two levels of branch history*. Proc. 20th Int. Symp. Comp. Arch., May 1993.