Appears in the Proceedings of the 33rd Annual International Symposium on Microarchitecture

The Impact of Delay on the Design of Branch Predictors

Daniel A. Jiménez   Stephen W. Keckler   Calvin Lin
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
{djimenez,skeckler,lin}@cs.utexas.edu

Abstract

Modern microprocessors employ increasingly complicated branch predictors to achieve instruction fetch bandwidth that is sufficient for wide out-of-order execution cores. While existing predictors can still be accessed in a single clock cycle, recent studies show that slower wires and faster clock rates will require multi-cycle access times to large on-chip structures, such as branch prediction tables. Thus, future branch predictors must consider not only area and accuracy, but also delay. This paper explores these tradeoffs in designing branch predictors and shows that increased accuracy alone cannot overcome the penalties in delay that arise with larger predictor structures. We evaluate three schemes for accommodating delay: a caching *approach, an* overriding *approach, and a* cascading *lookahead approach. While we use a common branch predictor,* gshare*, as the prediction component, these schemes can be constructed using most types of predictors.*

1 Introduction

Accurate branch prediction is essential to sustaining high IPC in pipelined microprocessors. Until now, the huge body of branch prediction research has focused on only two dimensions of the problem—area and accuracy—and has found that larger hardware budgets yield higher accuracy for two reasons: They allow longer history lengths, and they reduce *aliasing,* which occurs when two unrelated branches destructively share the same hardware branch prediction resources. Indeed, much of the recent work has focused on methods for reducing aliasing [19, 13, 12, 21, 5]. With growing chip capacities, the focus of the research community on area and accuracy has led to large elaborate predictors, some of which require 16K to 64K byte structures [7].

Recent studies, however, have shown that as feature sizes shrink, larger wire delays and smaller clock cycles will lead to multi-cycle access to large on-chip structures [1]. Thus, future branch predictors will need to consider a third dimension: delay. Figure 1 illustrates the problem of ignoring delay. Using an idealized delay of one cycle to access the pattern history table (PHT), the *gshare* predictor [13] sees improved IPC—due to improved prediction accuracy—as the size of the PHT is increased. By contrast, with an aggressive clock frequency (2GHz) and a realistic delay model for today's 180 nanometer technology, the curve drops off at 1KB where the PHT requires two cycles to access, and IPC drops significantly at 32KB where delay becomes three cycles. This problem will be exacerbated by the smaller process technologies of the future, as shown by the curve for 100nm technology, which drops to 3 cycles 8KB.

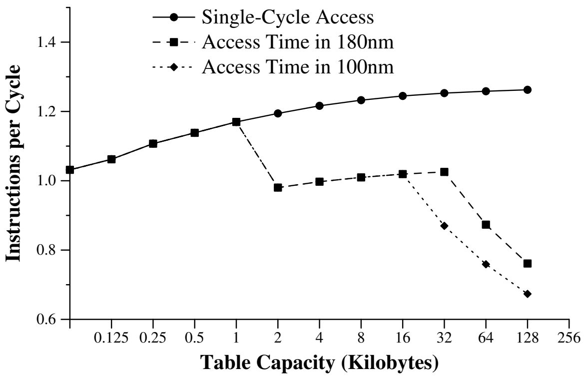

Figure 1: Instruction Throughput versus Capacity for the *gshare* predictor. Using idealized single-cycle access, IPC (and prediction accuracy) increases with increasing pattern history table capacity. Using realistic delay models, IPC drops when the delay is 2 cycles, and falls precipitously when the delay is 3 cycles.

This paper explores the tradeoffs in delay, area, and accuracy for the design of future branch predictors. We examine three approaches for dealing with delay in future process technologies: a two level caching scheme, an *overriding* scheme that allows a first prediction to be overturned by a more accurate second prediction, and a *cascading* lookahead scheme that exploits the time between branches to start reading prediction tables.

Each approach can be implemented with almost any two-level branch predictor as components. We use *gshare* as the basic prediction component because it is well-understood and often used as a standard for comparison. To calibrate our results with existing technology, we also simulate a hybrid predictor similar to that found in the Alpha 21264 [11] microprocessor, and we show how this hybrid predictor scales to future technologies.

This paper makes the following contributions.

- We show that delay in the predictor significantly erodes performance, so future branch prediction work must consider delay in their designs.
- We show that increasing delay to improve accuracy is never a good tradeoff.
- We show that there are approaches to branch prediction that can effectively use large structures with multi-cycle access times.
- We show that the overriding approach performs best and can improve IPC by 10% over *gshare* in 35nm technology at an aggressive clock rate.

This paper is organized as follows. Section 2 describes related work. Section 3 discusses the technological challenges that branch predictors face. Section 4 describes three approaches to dealing with multi-cycle delay, and Section 5 presents experimental results.

2 Related Work

Most recent research in dynamic branch prediction focuses on the two-level scheme of Yeh and Patt [23], which uses two-bit saturating counters to record the history of particular branches or branch patterns. As Sechrest, et al. showed [19], aliasing can limit the accuracy of branch predictors. A variety of techniques for reducing aliasing have been suggested [13, 12, 21, 5], and given sufficiently large predictor tables, many of these two-level achieve similar performance [5].

Lookahead branch prediction, including predicting multiple branches per cycle, has been suggested as a means for predicting branches that have not yet been presented to the predictor. One of the first lookahead branch predictors was proposed by Yeh, et al. [22] as the Multiple Branch Two-Level Adaptive Branch Predictor. This predictor uses the result of the first branch prediction to speculatively update the history register for a second branch prediction. No branch addresses are required since only the global history register is used to access the pattern history tables. Seznec, et al. improve on this idea by enhancing the BTB to enable the predictor to use the address of the current instruction block to perform prediction for the next instruction block [20]. This scheme enables the fetches to multiple blocks to be pipelined. Onder, Xu and Gupta propose a similar scheme in which predictions for an entire branch sequence are made all at once, and instruction fetch can continue unimpeded through the last branch [14].

Driesen and Hölze propose a “cascaded” predictor that dynamically filters easily predicted branches, relieving aliasing effects in the PHT [4]. Our work borrows the idea of cascading, but uses it to alleviate delay. Similarly, Evers describes the use of two PHTs with different history lengths and different access times, and suggests that the slower one can override the other [6]. The Alpha 21264 branch predictor uses the idea of overriding: the branch predictor can override the less accurate instruction cache line predictor, with a penalty of a single cycle, as opposed to the seven-cycle branch misprediction penalty [11].

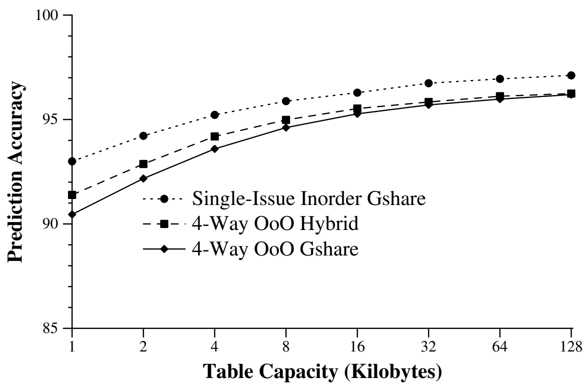

Figure 2: Prediction Accuracy versus Pattern History Table Capacity for our benchmarks. As the capacity increases, so does the prediction accuracy. Accuracy is worse for a 4-way out-of-order machine because the branch history is not always updated in time for the next prediction.

Of course, the real goal in these strategies is to improve instruction fetch bandwidth and preferably take branch prediction off the critical path. Recent research has focused on trace caches as a mechanism to capture a long stream of sequential instructions that can be easily fetched at peak bandwidth [18, 15]. Branch prediction guides the trace selection in the instruction fetch engine, at times predicting multiple branches per cycle. A more radical approach is the Fetch Target Buffer (FTB) proposed by Reinman, et al. [16]. The FTB stores the addresses of predicted blocks of instructions and is designed as a two level cache for fast access and accurate block prediction. Like our study, Reinman, et al. consider technology constraints in the design of the FTB. Frameworks like the FTB can benefit by using delay-sensitive branch prediction strategies as their branch prediction components.

3 Challenges for Branch Prediction

This section discusses the near term technological trends in fabrication technologies that branch predictor designers must confront. We first discuss the tradeoffs between accuracy and delay, explaining why delay is becoming increasingly significant. We then explain why large tables lead to large delays. Together, these observations frame our search for the latency-sensitive predictors that we discuss in the next section.

3.1 Predictor Delay vs. Accuracy

Branch prediction accuracy increases with the amount of memory allocated to the branch prediction table. Figure 2 shows the accuracy of the *gshare* branch predictor on several benchmarks (see Table 2 for a list of the benchmarks used in this study) as the prediction table capacity is increased, both in a sequential in-order machine and a 4-way out-of-order machine simulated with SimpleScalar [2]. The graph also shows the accuracy of a hybrid predictor similar to that in

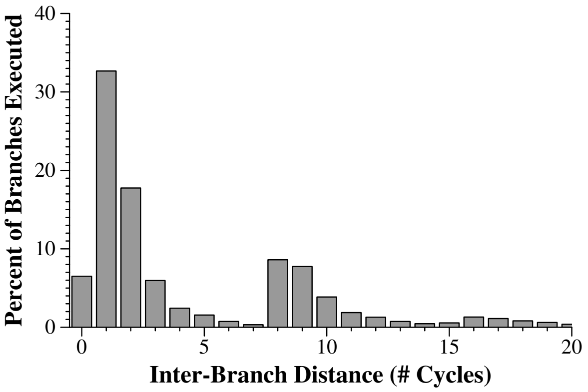

Figure 3: Histogram of average inter-branch latencies, measured in cycles, between prediction requests, for the SPEC 2000 integer benchmarks. Over 60% of the prediction requests occur at least one cycle after the previous request.

the Alpha 21264. We see that wide issue out-of-order execution has an important effect on prediction accuracy, increasing the misprediction rate roughly 25% over the single-issue in-order case because some predictions are demanded before the global pattern history register can be updated with the most recent branch outcomes. This effect also occurs with a single-issue in-order machine, but is much less pronounced since fewer branches can be in flight at the same time.

Similar graphs appear in most recent branch prediction papers. These graphs tacitly imply that branch prediction accuracy, and hence instructions-per-clock (IPC), can be improved by increasing the size of the prediction table. However, larger structures lead to larger access delays; worse, aggressively increasing clock rates (as the marketplace demands) increases the structure access time as measured in clock cycles.

Our studies show that it is almost never worth increasing the delay of a branch predictor for the sake of improved accuracy. For example, Figure 1 shows that as we increase the capacity of the tables in *gshare*, we increase delay and decrease IPC. This effect can be explained with the following equation which roughly approximates the cost C of executing a branch instruction:

$$C = d + (r \times p)$$

where d is the delay of branch predictor, r is the misprediction rate, and p is the misprediction penalty. While the delay d may not always be on the critical path of the pipeline, increasing d will reduce the instruction fetch bandwidth to the execution cores. Because misprediction rates tend to be close to 10%, changes in d have a larger impact than small changes in r.

3.2 Branch Frequency

A program's control behavior is based not only on the predictability of its branches, but also on the branch frequency. If branch prediction is required on every clock cycle, any delay in branch prediction will substantially slow the instruction fetch rate. However, if branches are widely spaced, then branch prediction latency will have less impact on performance. We use SimpleScalar to measure the average branch frequency in ten SPEC 2000 integer benchmarks on a 4-way out-of-order machine configuration. The results in Figure 3 show that 61% of the branches had at least one unused cycle between predictions. The unused cycles provide additional time to predict future branches.

Gate (nm)	16FO4 Clk f_{16} (GHz)	10FO4 Clk f_{10} (GHz)
250	0.70	1.12
180	0.96	1.54
130	1.33	2.13
100	1.74	2.78
70	2.48	3.97
50	3.47	5.55
35	4.96	7.94

Table 1: Projected clock rates using FO4 Clock scaling.

3.3 Technology Scaling

Branch predictors, like other microarchitecture structures, are affected by two technology scaling trends. At smaller feature sizes, wire delay grows in significance relative to transistor speeds and can affect the latency of the fetch engine and the branch predictor. Furthermore, microprocessor designers continue to aggressively increase the clock rates, outstripping the speed improvements achieved by transistors that have smaller gate lengths in each successive technology [1]. These faster clocks exacerbate the tradeoff between capacity and delay in microprocessor components.

To account for accelerating clock rates, we use a technology independent metric, the *fanout-of-four* (FO4) delay metric, to measure clock period [9]. One FO4 delay is the time for an inverter to drive 4 copies of itself. Reasonable models show that under typical conditions, the FO4 delay, measured in picoseconds, is equal to $360 \times L_{drawn}$, where L_{drawn} is the minimum gate length for a technology, measured in microns. The number of FO4 delays in a clock period is an indicator of the number of levels of logic in a pipeline stage. In this paper, we examine two edges of the clock scaling envelope: f_{10}, which corresponds to a clock period of 10 FO4 delays, and f_{16} corresponding to 16 FO4 delays. Table 1 lists the technologies that we consider in this paper and the clock rates that result from aggressive (f_{10}) and conservative (f_{16}) scaling.

We base our estimates of branch predictor delay on the access time of the memory-oriented structures such as the pattern history table (PHT). To model PHT delay, we use the methodology described by Agarwal, et al. [1], which augments the ECacti cache delay modeling tool [17] with scaled technology parameters. We convert the access time produced by the augmented ECacti model into cycles, according to both the f_{10} and f_{16} clock scaling strategies. As shown in Figure 4, with the aggressive f_{10} clock, only small tables of 1024 entries can be accessed in a single cycle, and at 35nm, only 512 entries can be accessed in one cycle. Accepting a 2 or 3 cycle delay increases the capacity to 16K and 64K entries, respectively. Using the conservative f_{16} clock rate,

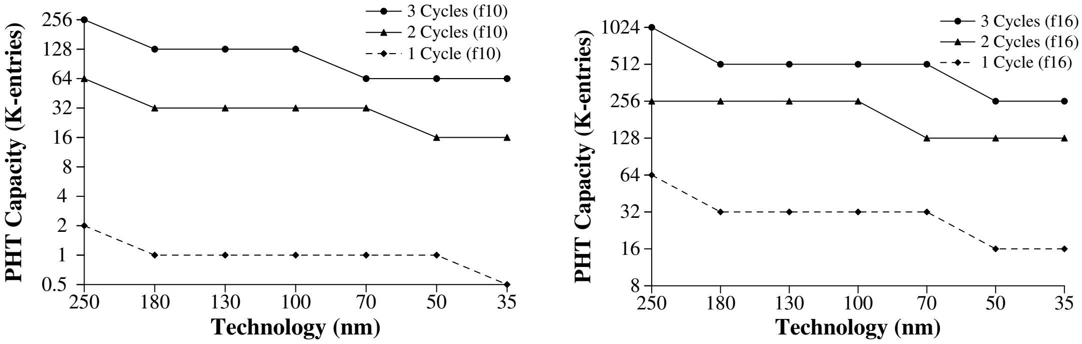

Figure 4: Pattern History Table capacity and access latency across technologies at agressive (f_{10}) and conservative (f_{16}) clock frequencies .

much larger structures can be used, ranging from 16K to 512K, as the access latency grows from 1 to 3 cycles.

As a consequence of technology and clock scaling, the challenge for the microarchitect is to achieve high accuracy in branch predictors whose table sizes are limited. Thus, branch predictors cannot be evaluated solely on prediction accuracy. Latency must be taken into account as the branch predictor will often reside on the critical path for the execution of the program. However, some of the latency of branch prediction can be hidden if spaces between branches can be found. The remainder of this paper examines techniques for achieving high prediction accuracy while minimizing prediction latency for future process technologies.

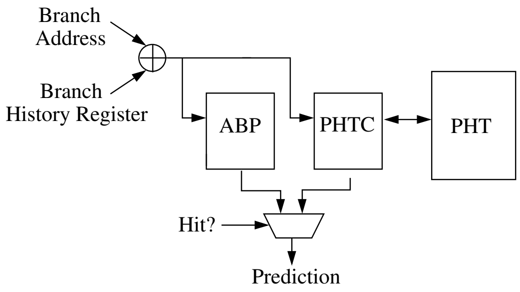

Figure 5: Caching Branch Predictor

4 Latency Sensitive Branch Predictors

In this section we describe three ways to configure branch predictors to increase accuracy in the face of increasing latency. These techniques all have a common theme: a small table is used to provide quick prediction, and a large table is used to provide higher accuracy. The techniques are appropriate when standard techniques for branch prediction might exceed one cycle, and are general techniques that can be applied to most prediction algorithms. We assume that the branch target buffer (BTB) is kept at a constant capacity and access time. While this is not realistic because of technology and clock rate impact on BTB capacity, it allows us to focus solely on the scaling of the branch predictor. Similar strategies can be applied to the BTB.

4.1 Caching Prediction Tables

The first strategy to combat the long latency of large branch prediction tables is to build a small cache of branch prediction table entries. This allows us to realize the benefits of reduced aliasing and increased history length without the added latency of the large table, since the cache will work in one cycle. Figure 5 shows the organization of the *gshare* predictor augmented by a cache. The branch history and branch address are hashed using the XOR gate, and the resulting address is sent to both the pattern history table cache (PHTC) and the pattern history table (PHT). The PHT consists of 2-bit saturating counters, with the number of counters equal to the number of combinations of addresses produced by the hash function. The PHTC caches a subset of those counters in a smaller table that can be accessed more quickly than the PHT. If the correct counter is found in the PHTC, then the prediction can be made immediately. If a miss in the PHTC occurs, then the PHT must be consulted to find the correct saturating counter. Like traditional caches, an entry in the PHTC is replaced with the correct counter from the PHT. When the branch direction is determined during a later stage of the execution pipeline, the counters in the PHT and PHTC are updated to reflect the correct or incorrect prediction of that branch.

If a PHTC miss occurs, the wait for the correct prediction from the PHT will delay instruction fetch and will degrade overall performance. Two alternatives can be used to prevent this additional delay. The prediction produced by the PHTC, albeit for the wrong branch, can be used. We instead build a small auxiliary branch predictor (ABP) that can be accessed at the same time as the PHTC. If the PHTC misses, then the result from the ABP is used. The accuracy of this hybrid-like predictor is a function of the capacities of the subsidiary predictors and the ability of the PHTC to capture the locality of branch instructions.

4.2 Cascading Lookahead Branch Prediction

Lookahead branch prediction has been proposed as a mechanism to increase fetch bandwidth by generating addresses for future branches [22, 20]. The same technique can be applied

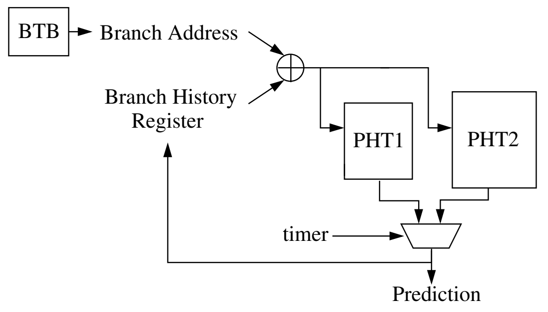

Figure 6: Cascading Branch Predictor

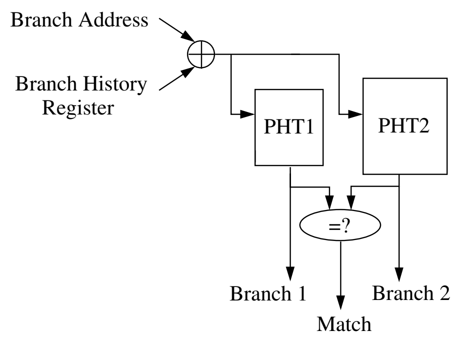

Figure 7: Overriding Branch Predictor

to reduce the impact of longer latency branch predictors. If the branch predictor is not needed on every cycle, then natural spacing between branches can be used to perform a prediction for the next branch that is likely to arrive. Thus, if branches are spaced so that the predictor is accessed only every other cycle, the predictor can have a two cycle latency without introducing additional delay.

The *gshare* predictor can be adapted to look one branch ahead. While *gshare* uses the branch history register and branch address to compute the PHT address, the lookahead predictor uses the predicted history and predicted branch target address. The predicted history is computed by appending the prediction of the most recent branch to the branch history register. The predicted branch target address is taken directly from the BTB as a result of the previous branch prediction. As a consequence, this scheme relies on the accuracy of the BTB. If the prediction can complete before the next branch arrives at the predictor, prediction is instantaneous. However, if the prediction requires multiple cycles (due to a large table) and the next branch arrives before the prediction is complete, the instruction fetch engine stalls.

Cascading lookahead branch prediction implements a series of tables of ascending size and latency. Figure 6 shows a two-level cascading predictor. Like a lookahead predictor, the next prediction is based on the last prediction and the last predicted branch target. Prediction is begun simultaneously on both levels of the cascading predictor. If the latency to the next branch to be predicted is large, then the prediction from the second level table is selected. If the next branch arrives before the second level table can complete its access, then the prediction from the first level table is used.

The combination of a small first level table and a larger second level table can provide high aggregate accuracy with low latency. However, the utility of the larger table depends on its access time and the inter-branch latency. If branches occur extremely frequently, the second level of the cascade will not be used. The cascading design can be trivially extended to more than two levels. Furthermore, hybrid predictors of varying latencies can be incorporated into the cascading strategy. In our description above, the logic to select which prediction to use is based only on the arrival time of the next branch. More complicated selectors could trade off latency versus accuracy by predicting which of many predictions is best for the subsequent branch.

Benchmark	Description
`164.gzip`	LZ77 compression
`175.vpr`	Place and route for FPGAs
`176.gcc`	C compiler
`181.mcf`	Minimum cost network flow solver
`197.parser`	Natural language processing
`253.perlbmk`	Perl
`254.gap`	Computational group theory
`255.vortex`	Database
`256.bzip2`	Block-sorting compression
`300.twolf`	Place and route

Table 2: Subset of the SPEC 2000 integer benchmark suite.

4.3 Overriding Branch Predictor

An overriding branch predictor (Figure 7) provides two predictions. The first prediction comes from a fast PHT (PHT1), and the second prediction comes from a slower, but more accurate PHT (PHT2). When branch prediction is requested, the first prediction is used and acted upon while the second prediction is still being made. If the second prediction differs from the first prediction, the actions taken based on the first prediction are squashed and instructions are fetched using the second prediction; thus, the second predictor overrides the first predictor. For the overriding scheme, we assume that the penalty of restarting an overridden fetch is equal to the delay of PHT2. A similar technique is used in the Alpha 21264, in which the branch predictor, whose results become known only in the second stage of the pipeline, can override the less accurate instruction cache line predictor [11] at the cost of a single stall cycle. We assume the predictor is pipelined such that no branch needs to wait for the completion of a PHT2 lookup for a previous branch.

	Capacity (bits)	# entries	Bits/entry	Ports
BTB	48K	512	96	1
Reorder buffer	8K	64	128	8
Issue window	800/320	20	56	8
Integer RF	5K	80	64	10
FP RF	5.6K	72	80	10
L1 I-Cache	512K	1K	512	1
L1 D-Cache	512K	1K	512	2
L2 Cache	16M	16K	1024	2
I-TLB	14K	128	112	1
D-TLB	14K	128	112	2

Table 3: Parameters used for the simulations, similar to the Alpha 21264.

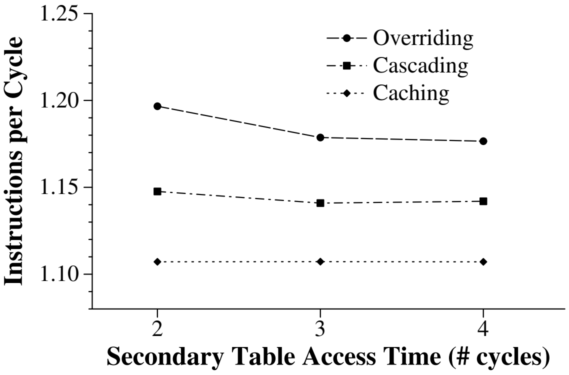

Figure 8: IPC at 100nm for various configurations of primary and secondary structures in the caching, overriding, and cascading predictors at the f_{10} clock rate.

5 Results and Evaluation

In this section we evaluate the three latency sensitive branch predictors and compare them to *gshare* across a spectrum of process technologies. As displayed in Table 2, we use ten SPEC 2000 integer benchmarks for our simulation. We simulate the different prediction strategies described above using delay estimates at seven process technologies ranging from 250nm to 35nm. We simulate the benchmarks using the SimpleScalar out-of-order simulator and PISA instruction set, configured with parameters similar to those for the Alpha 21264; the simulator is a modified version of the one used by Agarwal et. al. [1]. Each simulation runs for 500 million instructions or until the application terminates, whichever comes first. In the simulations, the global pattern history register is updated speculatively and backed up on a mispredict, while updates to the PHTs are done when the updating branch commits.

Since we are focusing solely on the branch predictor, we keep the other structure sizes constant at values shown in Table 3. Our main results use the aggressive f_{10} clock rate, which emphasizes the scaling difficulties of branch predictor structures. We also report results for the more conservative f_{16} clock. Although f_8 was used in the original technology scaling work [1], we choose f_{10} as our aggressive clock rate because our hybrid predictor is unworkable at the f_8 clock rate.

For each process technology, we configure the simulator with the largest branch prediction structures (predictor tables, cache, etc.) reachable at the given number of cycles allocated to branch prediction. The structure sizes are obtained using the modified version of ECacti described in Section 3. For each benchmark we measure IPC, aggregate branch predictor accuracy, and other statistics related to the branch prediction schemes. Aggregate branch prediction performance is computed as the arithmetic mean over the benchmarks. Note that the capacity of each structure is set by its access time, rather than any chip area limitation. With smaller feature sizes, this assumption is fair, as the amount of effective chip area is far larger than is reachable in the number of cycles we consider.

5.1 Predictor Configuration

For each predictor, we consider several configurations of structure capacity and latency in search of the best configuration at each technology generation. Figure 8 shows the results of these experiments for the caching and cascading predictors at 100nm. In the caching predictor, the two structures are the PHTC and the PHT, while in the overriding and cascading predictor the two structures are the PHT1 and PHT2. As the secondary structure access times increase, the resulting IPC is slightly worse for the overriding predictor and slightly better for the cascading predictor. The size of the secondary structure for the caching predictor makes little difference in performance. The rest of our results are reported using the best configurations found for each prediction technique.

Each *gshare* component of the various predictors uses the maximum history length, e.g., if a *gshare* predictor has 1024 entries, then the maximum history length is $log_2 1024 = 10$. Studies have shown that the using the maximum history length does not always yield the best accuracy [13, 8], so we empirically identify the best history length for *gshare* at each hardware budget. We find that, for our PISA instruction set, branch predictor configurations, and benchmarks, the best history length is always the maximum.

In the caching predictor, we varied the latency of the PHT from 2 to 4 cycles, keeping the PHTC at a 1-cycle access time. Note that increasing the latency of each table also increases its capacity.

For the cascading and overriding predictors, we keep access to the primary PHT at one cycle while varying access to the secondary PHT from from 2 to 4 cycles. Increasing the second level (PHT2) latency reduces IPC slightly for the overriding predictor, but increases IPC slightly for the cascading predictor.

The best configurations for the caching predictor at the f_{10} clock rate can be seen in Table 4. The PHTC has an unusually small number of entries compared with the other structures. Unlike a normal cache that has large cache lines, our caching predictor requires many times more tag bits than data bits. The extra wire length involved in accessing the tag

Technology (nm)	ABP Delay	ABP Entries	PHTC Entries	PHT Delay	PHT Entries
250	1	2K	512	2	64K
180	1	1K	256	2	32K
130	1	1K	256	3	128K
100	1	1K	256	4	256K
70	1	1K	256	2	32K
50	1	1K	256	2	16K
35	1	512	128	2	16K

Table 4: The best configurations of the ABP and PHT table sizes, as well as number of PHTC for the caching predictor at each technology for the f_{10} clock rate.

Technology (nm)	PHT1 Delay	PHT1 Entries	PHT2 Delay	PHT2 Entries
250	1	2K	2	64K
180	1	1K	3	128K
130	1	1K	2	32K
100	1	1K	2	32K
70	1	1K	3	64K
50	1	1K	3	64K
35	1	512	2	16K

Table 5: The best configurations of the PHT1 and PHT2 for the cascading and overriding predictors at each technology for the f_{10} clock rate.

bits severely restricts the number of cache entries, limiting the effectiveness of this scheme. Other prediction components in which the size of the basic prediction element is large with respect to the number of tag bits, such as the perceptron predictor [10], may be more amenable to a caching scheme.

The best configurations for the cascading predictor at the f_{10} clock rate are shown in Table 5. The best configurations for the overriding predictor are identical to those of the cascading predictor, since the two predictors have much the same architecture and differ only in their policy of when and whether to use the second-level PHT. Indeed, the stream of updates to the PHT1 and PHT2 structures should be the same in both overriding and cascading predictors; the only difference is that the overriding predictor always uses the PHT2 prediction, while the cascading predictor only uses the PHT2 prediction when it has enough time.

The best configurations for each predictor at the more conservative f_{16} clock rate (not shown) allow larger tables and single-cycle access for the hybrid predictor at every technology.

5.2 Structure Usage Rates

The simulations keep statistics on the rate at which the various structures are accessed. To explain the relative performance of each technique, Figure 9 shows the utility of each predictor. These statistics explain the relative performance of each technique. For the caching scheme, the the PHT is accessed for 7.5% of all branches, and this access results in

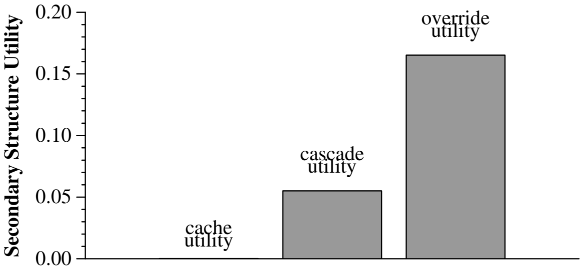

Figure 9: The utility of the secondary structures is highest in the overriding predictor and lowest in the caching predictor.

a prediction different from that of the ABP in 0.013% of all branches. This explains why the performance of the caching predictor is so similar to *gshare* by itself: it almost always relys on the ABP, and when it doesn't, the ABP and PHT almost always agree.

For the cascading scheme, the PHT2 structure is used for 45.6% of all branches, and its prediction differs from that of PHT1 for 5.5% of all branches, thus the second level table is useful for only 5.5% of branches.

For the overriding scheme, the frequency with which the predictions of PHT1 and PHT2 disagree, i.e., how often the more accurate predictor is used, is 16.5%, so the overriding scheme is the most useful of the predictors. These results are for 100nm technology; the statistics are similar across all technologies.

5.3 Hybrid Predictor

To demonstrate the effect of delay on predictors more complex than *gshare*, and calibrate our clock estimates with a real-world processor, we simulate a hybrid predictor similar to the branch predictor of the Alpha 21264. We report IPC and accuracy figures for this predictor along with our other results. This predictor maintains both global and local branch history information. The global pattern history register is used to index into a PHT while the branch address is used to index into a table of local histories, which is then used to index another PHT. A choice table is indexed by the global pattern history register. The global branch history is update speculatively, and all other tables are updated when the branch commits. We assume the lookups in the global PHT, local history table, and chooser table are all started at the same time, and the lookup in the local PHT occurs immediately after the local history register becomes available from the local table. The global PHT and chooser table have four times the capacity of the local PHT, and the local PHT and local histories table have the same number of entries.

This predictor closely resembles the Alpha predictor [11] with two exceptions: (1) on the Alpha, the prediction becomes available only in the second pipeline stage, and can override the first-stage line predictor, while our predictor operates in the first pipeline stage; and (2), we allow the capacity of our predictor to vary depending on the access times at

Technology (nm)	Delay	Local Entries	Global Entries
250	1	128	512
180	2	4K	16K
130	2	4K	16K
100	2	4K	16K
70	2	4K	16K
50	2	2K	8K
35	2	2K	8K

Table 6: The best configurations for the hybrid predictor at each technology for the f_{10} clock rate. Beyond 250nm technology, the predictor simply can't work in one cycle because of sequential lookups into the local history table and local PHT, so delay slips to two cycles, with a corresponding increase in table capacities.

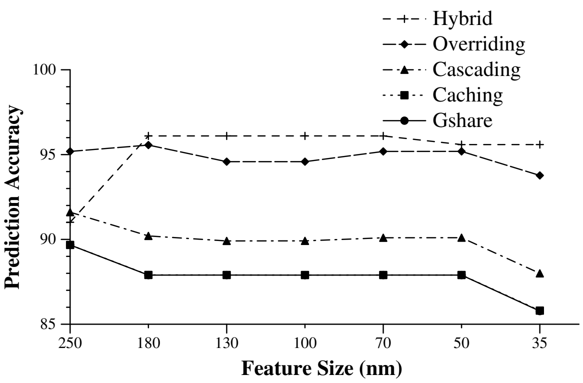

Figure 10: Accuracy vs. Technology for the five prediction strategies at f_{10}. The hybrid predictor is more accurate only because its best configuration consumes two cycles, allowing it to use large table.

the different clock rates and technologies.

The configurations for the hybrid predictor at the f_{10} clock rate are seen in Table 6. Unlike the *gshare*, this hybrid predictor requires two table accesses: first in the table of local histories, and then in the local PHT. Consequently, this hybrid predictor is much more sensitive to delay that prediction schemes that require only a single table access. In fact, at the aggressive f_{10} clock rates, we found that the table sizes are prohibitively small for single cycle predictor access at 180nm and smaller. Our original experiments used an f_8 clock rate, but that resulted in multi-cycle predictor latency at 250nm as well. Finally, using the table sizes of the Alpha 21264 in 250nm technology results in an access time of 1.55 ns, which corresponds to the published clock frequency of the Alpha of approximately 600MHz [11].

5.4 Aggressive Clocking

Figure 10 shows the accuracies of the best configurations of the various predictors at the f_{10} clock rates. As shown in the graph, accuracy tends to decrease with feature sizes, because the prediction table capacities decrease. The accuracy of the overriding predictor increases slightly from 100 to 70nm, since the best configuration for 70nm technology allows the PHT2 to take three cycles, while the best configuration in 100nm allows only two cycles. Likewise, the accuracy of the hybrid predictor increases from 250 to 180nm as the best configuration predicts in two cycles, allowing a larger area to be used. Of the schemes that can provide a prediction in a single cycle, the overriding predictor achieves the highest accuracy because it always uses larger second-level predictor, either because it agrees with or overrides the first-level predictor. The cascading predictor performs worse because it sometimes uses the less accurate first-level predictor, either because there are not enough cycles to use the second-level predictor, or because the branch target from the BTB is incorrectly predicted. Thus this predictor faces the challenge of branch misprediction as well as branch target misprediction. Finally, caching performs less well, not even exceeding the accuracy of a single level *gshare* predictor.

Of course, accuracy is not necessarily indicative of performance, particularly when prediction time is a variable. Figure 11 show the instruction throughput (IPC) for each of the configuration described above. The hybrid predictor, while achieving the best accuracy, reflects the lowest IPC at the smaller technologies, due to the access time increasing to two cycles. The rest of the predictors follow parallel trajectories with performance reflecting the overall accuracy of the predictor. Clearly, the overriding predictor, with it higher accuracy, is best for every process technology at the aggressive f_{10} clock rate.

5.5 Conservative Clocking

Figures 12 and 13 show accuracy and IPC of the same prediction schemes, but with different configurations for the more conservative f_{16} clock rate. At this clock rate, the accuracy of the predictors are very similar, since the first-level PHTs are larger. For instance, the cascading predictor can use a first-level PHT with 64K entries; having a larger second-level PHT2 does little to increase the accuracy of this predictor. Again, overriding achieves the highest accuracy, but the accuracy of hybrid decreases as feature size decreases. At the f_{16} clock rate, the hybrid scheme can be implemented in a single cycle, but the table sizes drop somewhat at 180nm and 35nm, causing a reduction in accuracy. The overall instruction throughput is similar, again reflecting the comparable accuracies of the different schemes.

That the IPC achieved by each scheme at f_{16} is no surprise. When the clock rate is set at a more conservative level, more time is available to all predictors and technology scaling is less critical. However, total performance is the product of the clock rate and the IPC. Thus the question is whether the IPC reduction at f_{10} outweighs the benefits of the faster clock rate. Note from Figures 11 and 13 that the IPC from the overriding scheme at f_{10} is only 3% less than that at the

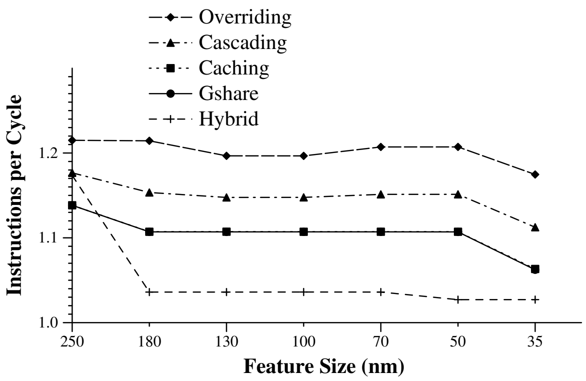

Figure 11: IPC vs. Technology for the five prediction strategies at f_{10}.

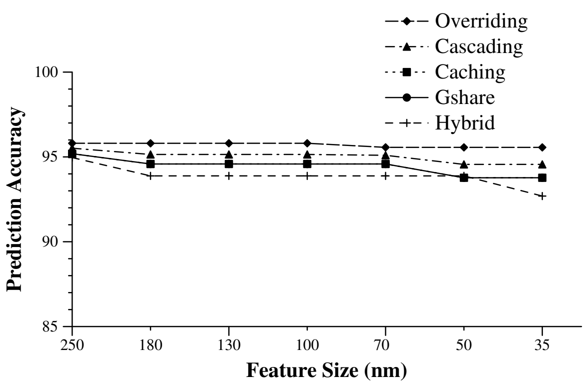

Figure 12: Accuracy vs. Technology for the five prediction strategies at f_{16}.

faster clock rate. Combined with a 1.6 times improvement in clock rate, overall performance will improve by using an aggressive clocking strategy. The benefits also exist with the other predictor schemes, but the benefits are somewhat less, due to larger degradation in IPC.

6 Conclusions

Until now, branch prediction design has focused on accuracy while ignoring delay. We have shown that as wire delays and clock rates increase, branch predictor designs that optimize for accuracy can have a negative impact on overall IPC. Thus, future branch predictor efficacy depends on both *accuracy* and *delay*, and researchers should account for both when reporting branch prediction results. According to our scalable models for branch predictor access time, today's predictors will not be accessible in a single cycle in sub-100nm technologies with aggressive clocking. In deep sub-micron technologies that are latency rather than capacity dominated, a branch predictor's area will become less important than its latency in the critical path.

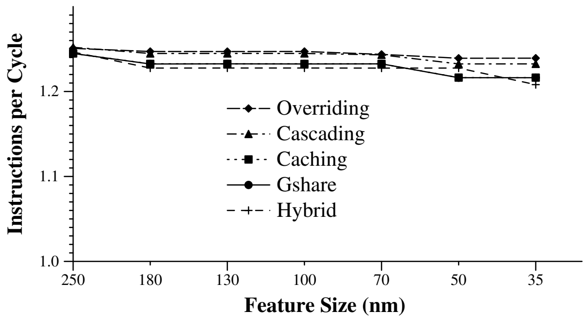

Figure 13: IPC vs. Technology for the five prediction strategies at f_{16}.

In this paper we have examined a number of alternative branch predictor architectures and evaluated them in the context of future process technologies. We found that a hybrid predictor is adequate until its latency exceeds one cycle, causing IPC to plummet. The predictor that caches a pattern history table (PHT) for *gshare* performs no better than *gshare* by itself. The tags needed to implement a caching scheme requires more bits than the cache itself, and limits both cache capacity and utility. The cascading lookahead predictor that uses the time in between branches to make predictions performs reasonably well at aggressive clock rates. The overriding predictor that allows a slow predictor to cancel the prediction of a faster, but less accurate predictor performs the best in our experiments.

To continue supplying a sufficient number of instructions to the execution core, future microarchitectures must move branch prediction latency off of the critical path. The schemes we present, particularly the cascading and overriding predictors, can be augmented by using something other than *gshare* as the primary or secondary predictor. We believe that the secondary predictor is the ideal place for a more complex and longer latency predictor, as it can be kept off of the critical path. Architectures such as the Fetch Target Buffer [16] are promising because they decouple the fetch engine from the execution engine. Other hardware alternatives include a more efficient branch predictor encoding such as that suggested by Jiménez and Lin [10], or multiple levels of cascaded predictors. The ideas of a cascading predictor and an overriding predictor can be combined, so that a late prediction from a second (or even third) PHT can override an earlier prediction; we believe this idea would outperform the overriding predictor by itself. Finally software may be able to assist further though branch classification [3] or through scheduling that increases the spacing of branches in the instruction stream.

7 Acknowledgements

We thank Vikas Agarwal for providing and modifying ECacti to enable better modeling of predictor structures, and Rajagopalan Desikan for his assistance in modeling predictors in SimpleScalar. We also thank the anonymous referees for their valuable suggestions. This research is supported by an IBM University Partnership Program award, NSF CAREER Grant ACI-9984660, DARPA Contract #F30602-97-1-0150, and ONR grant N00014-99-1-0402.

References

[1] V. Agarwal, M. Hrishikesh, S. W. Keckler, and D. Burger. Clock rate versus IPC: The end of the road for conventional microarchitectures. In *The 27th Annual International Symposium on Computer Architecture*, pages 248–259, June 2000.

[2] D. Burger and T. M. Austin. The simplescalar tool set version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin, June 1997.

[3] P.-Y. Chang and U. Banerjee. Branch classification: a new mechanism for improving branch predictor performance. In *Proceedings of the 27th International Symposium on Microarchitecture*, November 1994.

[4] K. Driesen and U. Hölze. The cascaded predictor: Economical and adaptive branch target prediction. In *Proceedings of the 31th International Symposium on Microarchitecture*, December 1998.

[5] A. Eden and T. Mudge. The YAGS branch prediction scheme. In *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, November 1998.

[6] M. Evers. *Improving Branch Prediction by Understanding Branch Behavior*. PhD thesis, University of Michigan, Department of Computer Science and Engineering, 2000.

[7] M. Evers, P.-Y. Chang, and Y. N. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In *Proceedings of the 23rd International Symposium on Computer Architecture*, May 1996.

[8] M. Evers, S. J. Patel, R. S. Chappell, and Y. N. Patt. An analysis of correlation and predictability: What makes two-level branch predictors work. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, July 1998.

[9] M. Horowitz, R. Ho, and K. Mai. The future of wires. In *Semiconductor Research Corporation Workshop on Interconnects for Systems on a Chip*, May 1999.

[10] D. A. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the Seventh International Symposium on High Performance Computer Architecture*, January 2001.

[11] R. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, March/April 1999.

[12] C.-C. Lee, C. Chen, and T. Mudge. The bi-mode branch predictor. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, November 1997.

[13] S. McFarling. Combining branch predictors. Technical Report TN-36m, Digital Western Research Laboratory, June 1993.

[14] S. Onder, J. Xu, and R. Gupta. Caching and predicting branch sequences for improved fetch effectiveness. In *International Conference on Parallel Architectures and Compilation Techniques*, October 1999.

[15] S. J. Patel, D. H. Friendly, and Y. N. Patt. Critical issues regarding the trace cache fetch mechanism. Technical Report CSE-TR-335-97, Department of Electrical Engineering and Computer Science, The University of Michigan, May 1997.

[16] G. Reinman, T. Austin, and B. Calder. A scalable front-end architecture for fast instruction delivery. In *Proceedings of the 26th International Symposium on Computer Architecture*, May 1999.

[17] G. Reinman and N. Jouppi. Extensions to cacti, 1999. Unpublished document.

[18] E. Rotenberg, S. Bennett, and J. E. Smith. Trace cache: A low latency approach to high bandwidth instruction fetching. In *Proceedings of the 29th International Symposium on Microarchitecture*, December 1996.

[19] S. Sechrest, C.-C. Lee, and T. Mudge. Correlation and aliasing in dynamic branch predictors. In *Proceedings of the 23rd International Symposium on Computer Architecture*, May 1999.

[20] A. Seznec, S. Jourdan, P. Sainrat, and P. Michaud. Multiple-block ahead branch predictors. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 116–127, October 1996.

[21] E. Sprangle, R. Chappell, M. Alsup, and Y. N. Patt. The Agree predictor: A mechanism for reducing negative branch history interference. In *Proceedings of the 24th International Symposium on Computer Architecture*, June 1997.

[22] T.-Y. Yeh, D. T. Marr, and Y. N. Patt. Increasing the instruction fetch rate via multiple branch prediction and a branch address cache. In *Proceedings of the 7th ACM Conference on Supercomputing*, pages 67–76, July 1993.

[23] T.-Y. Yeh and Y. N. Patt. Two-level adaptive branch prediction. In *Proceedings of the 24^{th} ACM/IEEE International Symposium on Microarchitecture*, November 1991.