Branch Meta-Prediction by Experts

Robert B. Gramacy
Dept. of Computer Science, UC Santa Cruz
Santa Cruz, CA 95064
rbgramacy@cse.ucsc.edu

Abstract

Predicting the direction of conditional branches with high accuracy has become critical to the success of new deeper and wider processor pipelines. However, choosing a branch predictor is hard because the best choice is workload-dependent. Meta- (or Hybrid-) predictors provide a rudimentary FSA-based heuristic for choosing between two predictors on-line. Such approaches have little or no theoretical motivation, and offer no performance guarantees. Recent developments in the Expert Framework provide explicit mechanisms for combining a pool of predictors and come with powerful theoretical results, bounding the loss the of master (meta-) predictor in terms of the loss of predictors in its pool. This paper shows how an arbitrary number branch predictors can be used as experts, and how the Expert Framework can be used to construct a *multi*-predictor. Our expert-based meta-predictors can achieve up to 62% fewer misses than the best predictor chosen in hindsight.

1 Introduction

Accurate branch prediction is critically linked to processor performance in deeply pipelined architectures. Branches can constitute a significant portion of instructions, and many other instructions depend on their computed target. Thus, poor predictions result in many discarded instructions. Any reduction in the rate of missed predictions translates into lower processor Cycles Per Instruction (CPI) and alleviates the need for complicated and hardware intensive *selective dual path execution* schemes.

Given a handful of prediction algorithms, choosing the best one can be a daunting task. Predictors of different flavors exploit varying criteria, and can have many tunable parameters. The current standard in dynamic branch prediction is to employ a *two-level* approach which trades-off global and per-address branch histories [YP92, PSR92, McF93]. Hierarchically indexing a branch address table (BAT), and then a pattern history table (PHT) results in the selection of a Finite State Automata (FSA). The FSA's state, and thus prediction, depends on direction of branches it predicted in the past. Despite the plethora of literature available which compare and contrast branch prediction algorithms, these studies have only one conclusive and unifying result: The best predictor is heavily workload-dependent. Even with perfect knowledge of the application, choosing from a pool of possible predictors off-line can be extremely difficult, and usually requires simulation.

Luckily, recent research has helped simplify things somewhat. Yeh *et. al.* [YP93], show that, with the exception of *gshare* [McF93], all two-level adaptive dynamic branch predictors essentially fall into one of nine categories. They demonstrate experimentally that, of

these nine categories, the "PAs" schemes are top performers on all but the SPEC [spe] `gcc` benchmark. However Gloy *et. al.* [GYCS96] argue that predictors like *gshare* and the nine of Yeh *et. al.* [YP93] can have varying success depending on how many cycles a workload executes in kernel or user code. Thus, choosing among only a handful of predictors is difficult.

Meta- (or Hybrid-) predictors [ERSM01, ECP96] have been proposed in order to take the burden of choosing the best branch predictor off of the architect. These approaches combine two or more predictors, by employing an extra (meta-set) of FSAs. Besides the obvious intuition behind why a hybrid approach might be successful, the only theoretical justification for it is a so-called reduction in *aliasing*. Apparently, aliasing reduction is "paramount" to prediction accuracy [ERSM01]. Such justification seems speculative at best. Moreover, this technique does not scale well beyond two predictors.

This paper proposes a hybrid-predictor with a strong theoretical motivation. The Expert Framework from on-line learning [HW95, CBFH$^+$97, LW94, BW01] gives a mechanism for monitoring the success of a pool of experts (predictors) and of combining their predictions. It comes with performance guarantees, bounding the accuracy of meta- (or master-) predictors in terms of the best predictor(s) chosen in hindsight. The framework is only marginally more complex than FSA-based schemes and gracefully scales to an arbitrary number of predictors.

2 Review & Related Work

This section surveys two-level dynamic branch prediction algorithms, and the Expert Framework from on-line learning. This is not the first application of Machine Learning to branch prediction. Loh *et. al* [aDSH02] give an approach which is loosely based on the WEIGHTED MAJORITY algorithm [LW94]. Jiménes and Lin [JC00] give an approach which uses Perceptrons.

2.1 Two-Level Adaptive Dynamic Branch Prediction

Table 1, taken from Yeh *et. al.* [YP93] outlines the dynamic branch prediction algorithms used in this paper. The table gives a brief description of each predictor and a reference. Both Gloy *et. al* [GYCS96] and Yeh *et. al.* [YP93] provide nice pictorial descriptions of these predictors, showing how the hierarchical tables are indexed (not duplicated here). Particular table specifications facilitate a trade off between global and per-address branch history (taken/not-taken) leading, finally, to a choice of FSA whose state is used for prediction. The exact semantics of the algorithm(s) executed by a particular two-level scheme can be complicated. The following discussion is included to give the underlying themes, aiming to bring out some high level similarities and differences. Details are left to the references.

Two-level predictors juggle per-branch history and global/local branch history to index into a first-level branch address table (BAT). Next, the branch address may or may not be used again to index from the BAT into a second-level pattern history table (PHT) table containing FSAs. One-level schemes (like `2bc`) can be thought of as two-level predictors which have an empty second level– where only the branch address is used. Global history schemes (`GAs`) use a BAT with only one shift register; per-address history schemes (`PAs`) use the branch address to select an entry in a larger BAT. Set-wise history schemes (`SAs`) behave similarly, but partition branch addresses into sets which cover the BAT, PHT, or both. In the limit, finely-granulated partitions lead to `PAs` and large partitions lead to `GAs`, etc.

The predictors in Table 1 can essentially be characterized by the number of bits used for the shift registers in each entry of the BAT; the number of FSAs used at the second level (PHT); the mapping of branch addresses into the BAT; and the branch histories and addresses which jointly index the PHT, where the actual FSAs reside. Small tables correlate branches

Variation	Reference	Description
Two-Level Adaptive		
GAg	[YP92]	Global Adaptive, with one global PHT.
GAs	[PSR92]	Global Adaptive, with per-set PHTs.
GAp	[PSR92]	Global Adaptive, with per-address PHTs.
PAg	[YP91]	Per-address Adaptive, with one global PHT.
PAs	[YP93]	Per-address Adaptive, with per-set PHTs.
PAp	[YP92]	Per-address Adaptive, with per-address PHTs.
SAg	[YP93]	Per-Set Adaptive, with one global PHT.
SAs	[YP93]	Per-Set Adaptive, with per-set PHTs.
SAp	[YP93]	Per-Set Adaptive, with per-address PHTs.
One-Level Adaptive		
gshare	[McF93]	Global history xor-ed, with per-address BAT.
2bc		Per-address BAT.

Table 1: Several branch prediction algorithms with brief descriptions and references. PHT = Pattern History Table; BAT = Branch Address Table.

Variation	# BAT Entries	# PHT Entries	BAT→PHT
GA{g,s,p}	1	$\{2^W, 2^{S_a+W}, 2^{W+A}\}$	direct
PA{g,s}	$\{2^W, 2^{W+A}\}$	2^A	direct
PAp	2^{W+A}	2^{A+W}	direct
SA{g,s,a}	2^{S_b}	$\{2^W, 2^{S_a+W}, 2^{A+W}\}$	direct
gshare	1	2^W	xor w/ BAddr
2bc	0	2^A	direct

Table 2: Branch prediction algorithms given in terms of table sizes. W is the width (number of bits) of the shift registers in each entry of the BAT; A is the number of lower-order address bits used to index a table entry for each branch; S_b is a the number of sets partitioning branches in the BAT; S_a similarly for the PHT. BAT entries map directly into PHT entries through shift register indexed by the branch address of the BAT; except `gshare` which xor's the shift register with the lower order branch address bits.

globally, and large tables tailor predictors to branches on a more individual basis. Table 2 shows how particular choices of table sizes at each level determine the predictor.

2.2 Expert Framework

While machine learning approaches seem scarce in the architecture realm, the Expert Framework is finding its way into many systems-related on-line prediction and optimization problems. Helmbold *et. al.* [HLSS00] use the Expert Framework to determine the optimal disk-spindown time for a mobile computer. Blum *et. al.* [BBK99] applied the framework to paging. Ari *et. al.* [AAG+02] used it to choose caching policies for nodes in a distributed cache. Recently, Gramacy *et. al.* [GWBA03, Gra03] showed how the framework can be used to design a meta-caching strategy which outperforms the best policy in its pool.

Informally, $\mathbf{e} = \{e_1, \ldots, e_N\}$ *experts* are decision making or prediction automata. Learning in the *Expert Framework* [CBFH+97] proceeds in trials $t = 1, \ldots, T$, where each expert receives an instance $x_t \in X$, predicts $e_n(x_t) \in Y$, receives as feedback the true label $y_t \in Y$, and incurs a *loss* $L_{t,n} = L_t(e_n(x_t), y_t) \in [0, 1]$. Branch predictors can be viewed as experts where the instances X are branch addresses and the labels Y are the actual, resolved, direction: $\{0, 1\} = \{\text{Taken, Not-Taken}\}$. A *master algorithm* combines the

prediction of experts by maintaining a probabilistic weight $\mathbf{w} = \{w_1, \ldots, w_N\}$ for each expert, thus encoding its belief in each as a predictor. Weights are updated after each trial t by a sequence of *updates*:

1. Loss Update:

$$w_t, n^m = \frac{w_{t,n} e^{-\eta L_{t,n}}}{\text{norm.}}$$

2. Share Update:

$$\mathbf{w}_{t+1} = \sum_{q=0}^{t} \gamma_{t+1,q} \mathbf{w}_q^m$$

where $\eta > 0$ and $\sum_{q=0}^{t} \gamma_{t+1,q} = 1$. Poor experts are punished by the loss update. The famous WEIGHTED MAJORITY algorithm [LW94], and STATIC EXPERT algorithm [HW95] are specimens which employ a loss update like that of above, but do not employ share updates. Theorems in the literature bound the loss of these master algorithms (A) by the loss of the best expert:

$$L_{1,\ldots T,A} \leq \min_n \{L_{1,\ldots T,n} + c \log_n\}, \qquad \text{for constant } c.$$

The share update keeps currently poor expert's weights from becoming too small. Several flavors of share updates are examined in the literature [HW95, BW01], of varying complexity and effect. The loss of these master algorithms are bounded by the best partition (P) of trials into segment/expert pairs:

$$L_{1,\ldots T,A} \leq \min_P \{L_{1,\ldots,T,P}\} + O(\text{\# of bits to encode } P)$$

where $L_{1,\ldots,T,P}$ is the loss of partition P.

3 Multi-predictor & Setup

The SimpleScalar [BA97] tool set is used to simulate each dynamic branch prediction scheme in Table 1 on SPEC [spe] benchmarks. SimpleScalar provides a very flexible branch predictor module allowing the hierarchical specification of table sizes and mappings. A representative spectrum of possible two-level branch predictors can be instantiated without much effort. SimpleScalar also provides a primitive meta-predictor, which makes a nice template for implementing our own expert-based *multi*-predictor. The prefix *multi* distinguishes the Expert Framework-based hybrid predictor from the FSA-based meta-predictor, highlighting its ability to employ more than two baseline predictors.

Our analysis verifies that the best predictor is workload-dependent and shows that Expert Framework can adjust accordingly. To update expert weights we use FIXED-SHARE TO UNIFORM PAST [BW01] (FSUP). This update is flexible and responsive, resource efficient, and exploits our prior belief that only a small sub-pool of the predictors will be helpful. Multi-predictors will be compared against one-level, two-level, and meta-predictors which use an equivalent or larger numbers of bits.

By default, SimpleScalar's meta-predictor (`Comb`) uses a table of 1024 2-bit saturating counters (simple FSAs), indexed by branch address. Each 2-bit saturating counter selects either a one-level (`BiMod`) predictor, or a two-level predictor (`2lev`)– `GAg` by default. The Expert-based multi-predictor is represented by 1024 sets of weights, indexed in exactly the same way.[1] Accordingly, the overhead required for meta and multi predictors differs by $O(E)$ for E expert weights. Each set of 1024 expert weights is updated by FSUP. Only after the correct branch direction is realized are experts weights and predictors updated. When predicting a branch, the single expert with the maximum weight is used for prediction. This is contrary to the what the theory behind the Expert Framework says to do (namely: that *all* experts contribute by a weighted sum or inner product). The max is taken here for simplicity– again allowing re-use of infrastructure in SimpleScalar for choosing predictors.

[1] This allows us to reuse the code provided by SimpleScalar for selecting meta-predictors.

Next comes the question of how many, and which, branch predictors should be experts. More base predictors means more bits, and adding more predictors only makes sense if the individual predictor's decision criteria remain diverse. We seek a simple spanning of the breath of two-level predictors for use as experts. The selection criteria should depend the number of desired experts, E.

3.1 Base Predictors

Initial experiments show that a modestly-sized single-level predictor (`BiMod`) often outperforms naively parameterized two-level predictors of the same size. As the total size of the predictor increases, the accuracy of `BiMod` saturates[2]. Correlated (two-level) branch predictors (`2lev`) surpass `BiMod` in prediction accuracy as the total size of the predictors is allowed to grow large. However, two-level schemes can saturate as well. With this in mind, our choices of expert predictors and size of the resulting multi-predictor will be completely determined by the number of experts, E. `BiMod` is always our first expert, and its table size is 2^{E-1}. All other experts are two-level predictors containing a BAT with 2^{e-1} shift registers, each with $E-e$ bits, and a PHT with 2^{E-1} 2-bit predictors, for $e = 1, \ldots E-1$.[3] The mapping between BAT and PHT is the default mapping implemented in SimpleScalar (`xor` $= 0$). Our aim here is one of demonstrating the flexibility and potential advantages of the Expert Framework by making the fewest assumptions possible. Otherwise, this choice of experts is arbitrary. In a real system experts would be developed with more care.

Summarizing: We have 1024 sets of expert weights indexed by branch address for E experts (expt$[e]$, $e = 1, \ldots E-1$), constructed as follows:

expt$[0]$ = BiMod(2^{E-1})
for $e = 1, \ldots, E$ do
    expt$[e]$ = TwoLev(BAT.dim = $2^{e-1} \times (E-e)$, PHT.dim = 2^{E-1}).
end for

This leads to a multi-predictor with total size

$$\text{Bits(multi}[E]) = 2 \cdot 2^{E-1} + \sum_{e=1}^{E-1} (E-e)2^{e-1} + 2E \cdot 2^{E-1} = 2^E + E2^E - E - 1.$$

Each of 1024 multi-predictors maintain their own weights ($\mathbf{w}$) over these E experts. `BiMod` and `2lev` experts are implemented using the machinery provided in SimpleScalar's `bpred` module.

4 Comparators, Results, & Discussion

SPEC [spe] Instructional Benchmarks (`anagram`, `compress`, `gcc`, and `go`) are used to compare multi-predictors of ranging numbers of experts, $E = 4, \ldots, 24$, each requiring bits as few as 75 and as many as 419,430,375.[4] The success of each multi-predictor was measured by pitting it against comparator predictors of similar size. Only predictors implemented natively in SimpleScalar are used as comparators. The main challenge in developing competitive (two-level) predictors as comparators is deciding how to allocate bits to tables at each level. After all, our goal is twofold: a fair comparison, and demonstration of the power of the multi-predictor. A multi-predictor which uses a total of B bits is pitted against three comparator predictors, each allowed to use at least B bits. These comparators are more arbitrary than optimal.

[2] This happens when the table size is larger than what can be indexed by a full branch address.

[3] We assume that $E \geq 2$. Things are not very interesting otherwise.

[4] This is not counting the hardware required to implement the table of $1024 \times E$ expert weights.

- `BiMod`: One table of 2-bit saturating counters indexed by (binary) branch address. size(BiMod(B)) $= 2 \cdot 2^{\lceil \lg B \rceil - 1} = 2^{\lceil \lg B \rceil} \geq B$.
- `2lev`: B bits are naively partitioned into two groups, one for each table: roughly 1/4 (i.e. $2^{\lceil B/4 \rceil - 3}$) for the BAT's (default) 8-bit shift registers indexed by branch address, and 3/4 (i.e. $2^{\lceil 3B/4 \rceil - 1}$) for the PHT 2-bit predictors, jointly indexed by the the selected BAT shift register and branch address. size(2lev(B)) $= 8 \cdot 2^{\lceil \lg B/4 \rceil - 3} + 2 \cdot 2^{\lceil 3B/4 \rceil - 1} = 2^{\lceil \lg B/4 \rceil} + 2^{\lceil \lg 3B/4 \rceil} \geq B$.
- `Comb`: An FSA-based meta-predictor comprised of a `BiMod` predictor and a `2lev` predictor, with selection based on a 2-bit saturating counter. A *meta-table* (MT) of 1024 entries is used, and does not factor into the total size of the comparator. B bits are divided up roughly equally between the baseline predictors B1 (`Bimod`) and B2 (`2lev`). B1 and B2 are constructed as above (for $B/2$ bits).

$$\begin{aligned}\text{size(Comb}(B)) &= \text{size(BiMod}(B/2)) + \text{size(2lev}(B/2)) \\ &= 2 \cdot 2^{\lceil \lg B/2 \rceil - 1} + 8 \cdot 2^{\lceil \lg B/8 \rceil - 3} + 2 \cdot 2^{\lceil 3B/8 \rceil - 1} \\ &= 2^{\lceil \lg B/2 \rceil} + 2^{\lceil \lg B/8 \rceil} + 2^{\lceil 3B/8 \rceil} \geq B\end{aligned}$$

In code, the above three comparators are constructed as follows:

cmp[0] = BiMod($2^{\lceil \lg B \rceil} - 1$)
cmp[1] = TwoLev(BAT.dim = $2^{\lceil \lg B/4 \rceil - 3} \times 8$, PHT.dim = $2^{\lceil \lg 3B/4 \rceil - 1}$)
cmp[2] = Meta(MT.dim = 1024, BP.1 = BiMod($2^{\lceil \lg B/2 \rceil} - 1$),
BP.2 = TwoLev(BAT.dim = $2^{\lceil \lg B/8 \rceil - 3} \times 8$, PHT.dim = $2^{\lceil \lg 3B/8 \rceil - 1}$)).

Figure 1 shows how our Expert Framework-based multi-predictors, of varying size, measure up against the comparators outlined in the previous section. These plots show a number of interesting things. For all four benchmarks the multi-predictor out-performs its comparators when they are constricted to use only a small number of bits. Similar results are realized when the multi-predictor is allowed to use a modestly large to extremely large number of bits.

With the exception of the `anagram` benchmark (`ana`)– for which all but the `BiMod` comparator seems to saturate when given large number of bits– only the multi-predictor improves persistently as more bits are added. Predictors like `BiMod` and `2lev` cannot take advantage of large BATs and PHTs. `Comb` better exploits a large allotment of bits. However, even as its allotment gets large, `Comb`'s baseline predictors can themselves can become saturated. Our expert-based multi-predictor completely frees itself from this limitation by incrementally adding predictors and reorganizing parameters when given more bits.

For predictors of small to medium numbers of bits (512-2048), the difference in prediction accuracy between the multi-predictor and its comparators is small. Sometimes the `BiMod` and `Comb` comparators even beat the multi-predictor. This may be an artifact of how the experts are constructed for small E (e.g. $E = 5, \cdots, 12$). Deeper investigation showed that the `BiMod` baseline predictor is chosen by the multi-predictor 80% of the time in such cases. Perhaps allocating more bits to `BiMod` when bits are in limited supply might help.

When E is large ($\gg 10$), the multi-predictor's leverage is maximized. Our results show that $E = 24$ can provide up to a 70% decrease in miss-predictions over the *best* comparators. For the `comp` benchmark, Figure 1 indicates that larger E may lead to an even further increase in accuracy. Likewise the `gcc` and `go` benchmarks have a 50% and 62% decrease in miss rate, respectively, for $E = 24$. Thus the multi-predictor is a clever mechanism for

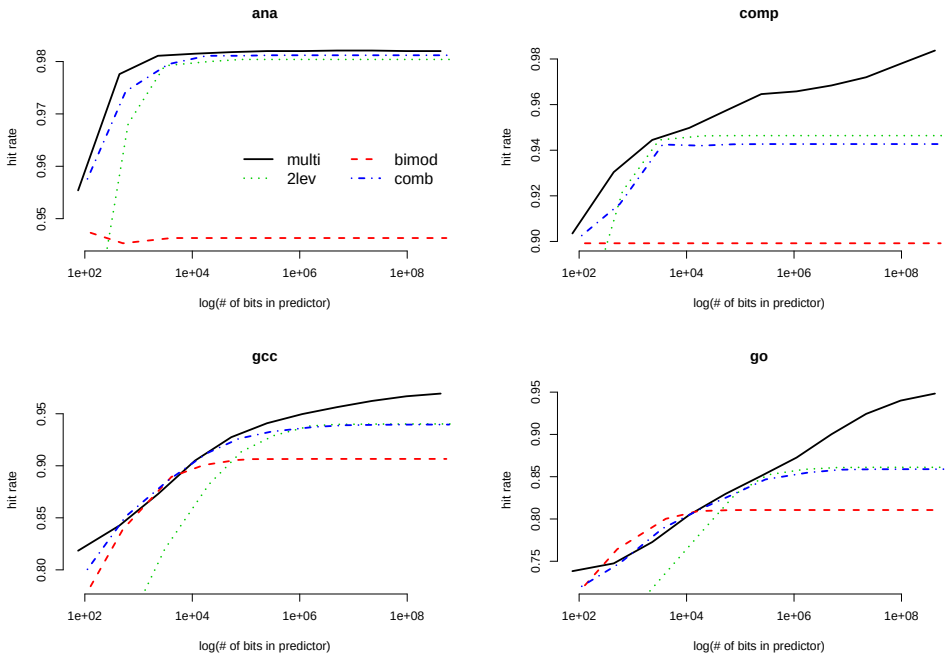

Figure 1: Multi-predictor results on the four instructional benchmarks from Spec, together with results for comparators outlined in Section 4. The x-axes range logarithmically over the total size (bits B) used by the predictors; the y-axes denotes the fraction of correctly predicted branches.

exploiting a surplus of prediction bits.[5] Results for smaller E are impressive as well, as can be also be seen from the figure.

5 Conclusions & Future Work

This paper outlines an automated framework for choosing dynamic branch predictors for any workload. The idea is not to choose a single predictor, but to choose a spectrum of baseline predictors, and let the Expert Framework from Machine Learning– and the wealth of theoretical results behind it– take it from there. We have provided the initial work in developing an analyzing such a framework, and have highlighted many of its benefits.

In many ways this work is still very preliminary. Evidence from many sources suggests that combing expert's branch predictions [aDSH02, Gra03] is far more effective than predicting with the max. Also, the spectra of appropriate baseline predictors needs more careful study. Comparators such as off-line optimal partitions of branches into segments should also be included. These partitions can usually be computed using dynamic programming [Gra03]. Finally, experiments should be done with different (e.g. smaller) numbers of multi-predictors (≤ 1024).

[5] Analyzing branch predictors in the limit, as the total size of the predictor(s) becomes large, is quite common in the literature. Our results show that the multi-predictor clearly holds its own in this comparison.

References

[AAG+02] I. Ari, A. Amer, R. Gramacy, E. L. Miller, S. A. Brandt, and D. D. E. Long. ACME: adaptive caching using multiple experts. In *Proceedings of the 2002 Workshop on Distributed Data and Structures (WDAS 2002)*. Carleton Scientific, 2002.

[aDSH02] G.H. Loh ad D. S. Henry. Predicting branches with fusion-based hybrid predictors. In *Proceedings in the IEEE International conference on Parallel Architectures and Compilation techniques*, pages 165–176, 2002.

[BA97] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

[BBK99] A. Blum, C. Burch, and A. Kalai. Finely-competitive paging. In *IEEE Symposium on Foundations of Computer Science*, pages 450–458, 1999.

[BW01] O. Bousquet and M. K. Warmuth. Tracking a small set of experts by mixing past posteriors. In *COLT/EuroCOLT*, pages 31–47, 2001.

[CBFH+97] N. Cesa-Bianchi, Y. Freund, D. Haussler, D. P. Helmbold, R. E. Schapire, and M. K. Warmuth. How to use expert advice. *Journal of the ACM*, 44(3):427–485, 1997.

[ECP96] M. Evers, P. Chang, and Y. N. Patt. Using hybrid branch predictors to improve branch prediction accuracy in the presence of context switches. In *ISCA*, pages 3–11, 1996.

[ERSM01] A. Eden, J. Ringenberg, S. Sparrow, and T. Mudge. Hybrid myths in branch prediction. In *Conf. on Information Systems Analysis and Synthesis (ISAS 2001)*, volume XIV, Comp.Sci. & Engineering, pages 74–81, July 2001.

[Gra03] R. B. Gramacy. Adaptive caching by experts. UC Santa Cruz, Masters Thesis, 2003.

[GWBA03] R. B. Gramacy, M. K. Warmuth, S. A. Brandt, and I. Ari. Adaptive caching by refetching. In *Advances in Neural Information Processing Systems (NIPS 2002)*. Morgan Kauffman, 2003.

[GYCS96] N. Gloy, C. Young, J. B. Chen, and M. D. Smith. An analysis of dynamic branch prediction schemes on system workloads. In *Proc. 23rd Annual Intl. Symp. on Computer Architecture*, pages 12–21, 1996.

[HLSS00] D. P. Helmbold, D. D. E. Long, T. L. Sconyers, and B. Sherrod. Adaptive disk spindown for mobile computers. *Mobile Networks and Applications*, 5(4):285–297, 2000.

[HW95] M. Herbster and M. K. Warmuth. Tracking the best expert. In *International Conference on Machine Learning*, pages 286–294, 1995.

[JC00] D. Jim and E. Calvin. Dynamic branch prediction with perceptrons, 2000.

[LW94] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994. An early version appeared in FOCS 89.

[McF93] S. McFarling. Combining Branch Predictors. Technical Report TN-36, June 1993.

[PSR92] S. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, pages 76–84. ACM Press, 1992.

[spe] The standard performance evaluation corporation. http://www.spec.org.

[YP91] T. Yeh and Y. N. Patt. Two-level adaptive training branch prediction. In *Proceedings of the 24th annual international symposium on Microarchitecture*, pages 51–61. ACM Press, 1991.

[YP92] T. Y. Yeh and Y. N. Patt. Alternative implementations of two-level adaptive branch prediction. In *Nineteenth International Symposium on Computer Architecture*, pages 124–134, Gold Coast, Australia, 1992. ACM and IEEE Computer Society.

[YP93] T. Yeh and Y. N. Patt. A comparison of dynamic branch predictors that use two levels of branch history. In *ISCA*, pages 257–266, 1993.